MASTER EN SCIENCES INFORMATIQUES

Système de recherche bibliographique pour chimiste organicien : introduction graphique et codage des molécules

Chaboteaux, Guy

Award date:
1987

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

SYSTEME DE RECHERCHE BIBLIOGRAPHIQUE POUR CHIMISTE ORGANICIEN : INTRODUCTION GRAPHIQUE ET CODAGE DES MOLECULES

Promoteur : M. NOIRHOMME

Mémoire présenté en vue de l'obtention
du grade de
Licencié et Maître en Informatique
par
Guy CHABOTEAUX

Année académique 1986-1987

Je tiens à remercier Madame Noirhomme qui a bien voulu accepter de diriger mon travail et m'a fait part de remarques intéressantes.

Ma gratitude s'adresse aussi à Monsieur Krief qui a assuré ma formation en chimie organique.

"Si nous désirons prendre les choses à leur commencement, il nous faut remonter à quelque trois milliards d'années en arrière au moment où la vie commençait à s'installer sur la terre. Notre jeune planète présentait alors toutes les conditions favorables à l'apparition de la vie telle que nous la connaissons : une température stable, ni trop élevée, ni trop basse, un apport d'énergie important provenant du soleil, une masse suffisante pour retenir une atmosphère et enfin le petit nombre d'éléments - carbone, hydrogène, oxygène et azote - à partir desquels toute matière vivante est formée. Ces quatre éléments représentent plus de 98 % de la matière des tissus vivants, ce qui est plutôt curieux si l'on songe que les éléments, qui constituent à eux seuls la presque totalité des composés organiques connus, sont des éléments rares de l'écorce terrestre où ils existent en proportion de 1 % ou même moins.

Comment la vie commença-t-elle ? De quelle manière des composés organiques complexes purent-ils se constituer à partir de ces atomes et des molécules simples qui existaient alors sur notre planète à peine vieille d'un milliard et demi d'années ? En 1923, un chimiste russe, A.I. Oparine, suggéra que les premières molécules organiques, ces "précurseurs" de la vie, prirent naissance dans un monde qui ne contenait que peu - voire même pas du tout - d'oxygène libre. L'atmosphère renfermait alors de la vapeur d'eau ainsi que du dioxyde de carbone, de l'azote, de l'ammoniac (NH_3) et du méthane (CH_4). La désintégration des substances radioactives terrestres venait ajouter son énergie dans ce creuset qu'embrasait un soleil générateur de nuages d'où tombaient des pluies diluviennes parmi l'éclatement incessant des éclairs. Suivant Oparine, ce fut dans ce chaos que les premières molécules organiques se formèrent et que la vie put commencer. Les gaz simples cités plus haut

étaient dissociés et leurs éléments constituants pouvaient alors se recombiner en composés plus complexes.

Au cours des années 1950, Stanley Miller entreprit des recherches en vue de confirmer les idées d'Oparine. Dans son laboratoire de l'Université de Chicago, il reconstitua, en mélangeant du méthane, de l'ammoniac, de l'eau de de l'hydrogène, une atmosphère identique à celle qui, selon Oparine, entourait la terre au moment de l'apparition de la vie. En faisant éclater dans ce mélange des étincelles des premiers temps, il put montrer que parmi d'autres molécules organiques, des acides aminés s'étaient formés. Ce résultat est capital puique toutes les protéines, qui sont les constituants majeurs de la matière vivante, résultent de la condensation d'amino-acides entre eux.

Les molécules organiques simples ainsi synthétisées pendant des millénaires se sont dissoutes dans les océans primitifs, où s'accumulèrent progresivement une grande variété de composés organiques. Toutefois, nous ne savons pas encore actuellement comment ces molécules organiques primitives évoluèrent pour constituer les cellules vivantes. Avec l'aide du temps, ce milieu complexe donna naissance, d'une manière ou d'une autre, à des agrégats moléculaires capables de croître et de se diviser en deux parties identiques d'une façon organisée, se donnant ainsi le pouvoir de se reproduire.

Une chose est certaine : tout au long de ce processus d'évolution et conjointement avec l'eau, la lumière solaire et un petit nombre d'éléments, l'atome de carbone a joué, et joue encore, un rôle tout à fait essentiel." [1]

TABLE DES MATIERES

INTRODUCTION

Certaines réactions chimiques, comme la fabrication de savon, la fermentation alcoolique, sont connues et utilisées depuis des siècles. Cependant, jusqu'il y a 200 ans, la chimie organique consistait essentiellement en la description des sources de matières premières animales et végétales disponibles (comme l'acide formique obtenu par distillation de fourmis !).

A cette époque, les chimistes pensaient que ces substances organiques ne pouvaient être produites que par des systèmes vivants auxquels ils associaient une "force vitale". Sans cette force, qui ne pouvait agir sans l'intervention d'un être vivant, la synthèse de ces produits ne pouvaient être possible. Cette idée fut cependant abandonnée lorsque l'urée, produit organique, fut synthétisée et identifiée comme telle en 1828 à partir de composés minéraux.

A la fin du 19ème siècle, plusieurs concepts importants, permettant une meilleure compréhension des structures des produits ainsi que de leur réactivité commençaient à apparaître. Depuis, la bibliothèque de réactions et produits organiques ne cesse de croître. La quatrième édition de "Dictionary of Organic Compounds" édité en cinq volumes en 1965 contient une brève description de plus de 40 000 composés organiques. Il existe des milliers de journaux qui publient des articles de chimie dont une soixantaine se restreignent à la chimie organique "pure". Certains sont même publiés toutes les semaines ou tous les quinze jours. [2]

Il devient dès lors intéressant de proposer aux chimistes organiciens un système capable de faciliter l'accès à toutes ces informations.

Les différents systèmes informatiques, opérationnels ou non, existant dans le domaine de la synthèse organique ont été présentés et commentés par Michel Bovesse [3] dans un mémoire défendu en 1985. Celui-ci proposait également une "solution graphique" pour construire une base de données en chimie organique. Ce travail fut ensuite repris dans le cadre d'un laboratoire [4] par Jean-Cristophe Gillet, Jean-Marc Trinon, Roland Noël et Thierry Huwart. Un des buts de ce travail est d'obtenir un système capable de tourner sur micro-ordinateurs. L'introduction des molécules se fera par dessin et sera réalisée avec les possibilités graphiques standards de ces appareils.

C'est également ce projet que nous allons poursuivre où nous allons essentiellement développer le mode d'introduction graphique des molécules ainsi que leur description.

CHAPITRE I

DESCRIPTION DES MOLECULES SUR BASE DE SOUS-STRUCTURES

1. Présentation du système existant

1.1. Utilisation de sous-structures

L'idée de base sur laquelle Michel Bovesse repose son système est la suivante :

> Pour le chimiste, une molécule est avant tout un ensemble de sous-structures réactionnelles pouvant être des groupes fonctionnels, des chaînes de carbones et des cycles.

Une molécule sera introduite sous la forme d'un dessin tridimentionnel. L'utilisateur disposera de l'ensemble de ces sous-structures prédessinées. Celles-ci pourront être "accrochées" via l'emploi de "prolongements" prédéfinis pour chacune d'elles. Cela nécessitera des opérations de translations et de rotations autour de trois axes.

Nous allons commencer par présenter quelques problèmes relevés par Jean-Cristophe Gillet, Jean-Marc Trinon, Roland Noël et Thierry Huwart lors de leur laboratoire.

Pour des raisons d'efficacité, il s'est avéré qu'une représentation en perspective des molécules n'était pas réalisable sur micro ordinateur, machine sur laquelle le système devra être implémenté. Cette décision est, en pratique, peu gênante dans la cadre de notre problème. En

effet, les chimistes utilisent le plus souvent des représentations planes des molécules sauf dans les cas où les réactions chimiques sont liées à la stéréochimie (disposition des groupements dans l'espace) des produits. Lorsque cela sera nécessaire, il sera beaucoup plus simple de spécifier par un symbôle quelconque la position relative des groupements de la molécule.

Ex. :

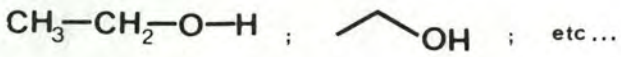
1.2. Problèmes des sous-structures

Les sous-structures peuvent être représentées en quatre grandes classes reprises ci-dessous :

- les groupes fonctionnels,
- les cycles,
- les chaînes carbonées,
- les atomes.

Pour chacune d'entre elles, il faudra choisir une représentation standard.

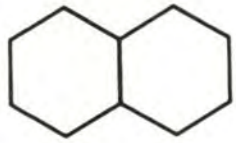
Ex. : un alcool peut être représenté de plusieurs façons :



Il sera cependant plus simple de ne proposer qu'une des représentations possibles dans le dictionnaire des sous-structures.

Les choses ne sont pas plus simples du côté de la représentation des cycles. En effet, toutes les liaisons formées par accrochage de sous-structures doivent être

formées via le recouvrement des prolongements. Comment alors décrire l'ensemble de cycles suivant



puisqu'il ne peut être construit à l'aide des prolongements des cycles représentés en traits plus gras dans le dessin ci-après :

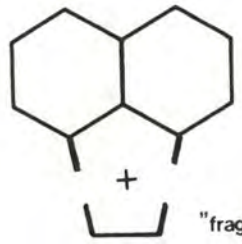
Il ne sera donc pas possible de n'utiliser ici que les sous-structures cycliques disponibles.

La seule solution, pour utiliser ici la notion de prolongements, semble être la création de "sous-structures spéciales" nécessaires à la jonction des cycles.

Ex. :

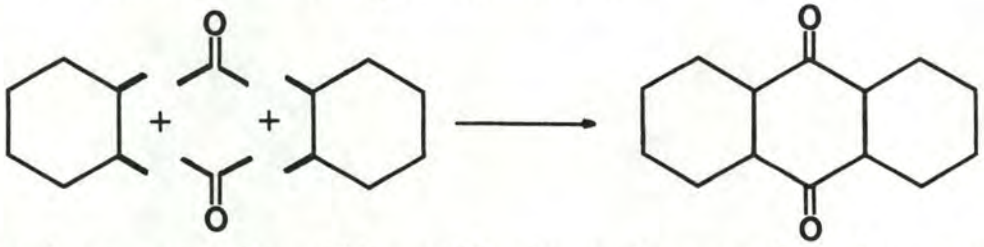
Il faudra prévoir de telles sous-structures pour toutes les tailles de cycle utilisées. Il sera de plus nécessaire d'en créer d'autres pour divers ensembles de cycles. Il n'est, en effet, pas encore possible d'obtenir la molécule suivante :

qui ne peut être construite que de la façon décrite ci-après à l'aide d'une nouvelle sous-structure :



Par ailleurs, ce nouveau type de sous-structures devra être "déformable" pour pouvoir s'adapter aux différentes tailles des cycles auxquelles elle sera liée.

Une fois ce problème résolu, il reste encore celui de l'introduction de fonctions chimiques sur un cycle comme c'est le cas dans l'exemple suivant :



Pourra-t-on par la suite effectuer des recherches sur le polycycle complet ?

Au niveau même du dessin, les problèmes ne sont pas si simples non plus. Nous avons déjà évoqué les sous-structures "déformables" et leurs problèmes. De plus, chaque sous-structure devra être apte à subir translations et rotations de manière à pouvoir être accrochée aux autres sous-structures de la molécule en construction. Pour chacune de celles-ci, il va donc falloir construire une table reprenant les coordonnées relatives de chaque atome et prolongement par rapport à un point privilégié de la sous-structure. Etant donné qu'il est peu probable de pouvoir réaliser un

dictionnaire de sous-structures complet, il faudra rechercher une méthode permettant d'en ajouter facilement.

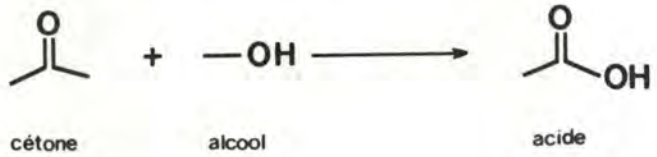
Cela nous amène à considérer la taille du dictionnaire. Celle-ci ne sera pas négligeable et la sélection d'un élément risque de ne pas être aisée. De fait, il ne sera pas possible d'afficher simultanément tous les choix possibles. Deux solutions sont alors envisageables :

- grouper les sous-structures par familles
- utiliser un dictionnaire sous forme d'un fascicule, le chimiste introduisant alors le numéro identifiant de la sous-structure nécessaire.

La deuxième méthode semble bien peu pratique. Quant à la première, si elle évite l'emploi d'un "élément séparé", elle ne sera malgré tout pas simple d'emploi. En effet, dans bien des cas, une "famille" ne pourra non plus être affichée entièrement (ce sera probablement le cas s'il faut plusieurs sous-structures par taille de cycle) d'où nécessité de défilement, de division des familles en sous-familles, etc... Cela reste donc bien lourd pour introduire une molécule si simple à dessiner sur papier !

Après l'évocation de ces problèmes mis en évidence par Jean-Cristophe Gillet, Jean-Marc Trinon, Roland Noël et Thierry Huwart dans leur travail, nous allons en relever personnellement quelques uns. Il s'agira notamment de problèmes au niveau des fonctions (qui sont normalement introduite sous forme de sous-structures uniques et indissociables). On peut en effet remarquer que la composition de plusieurs sous-structures risque de former une nouvelle sous-structure qu'il faudra donc reconnaître.

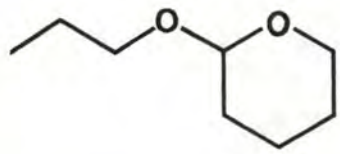
Ex. :



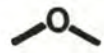
Ce qui donne ici deux méthodes de construction de la fonction acide, puisque celle-ci doit normalement exister comme sous-structure indissociable dans le dictionnaire. Et cet exemple est loin d'être unique !

Ce problème est malheureusement très difficile à résoudre, puisque dans certains cas de figures, il ne sera pas possible de procéder autrement.

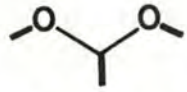
Ex. :



Comment construire cette molécule autrement qu'en utilisant un cycle modifié, la sous-structure



et enfin, une chaîne carbonée ? Or, la fonction intéressante de la molécule est la suivante :



et il est impossible de l'introduire à l'aide d'une sous-structure unique. Ce genre de problème apparaîtra souvent lorsque des cycles seront présents dans la molécule.

Pour terminer, nous allons revenir sur les "fragments de cycles" nécessaires pour la construction de certains ensembles de cycles. Nous avons vu qu'il sera nécessaire de pouvoir déformer ces fragments. Il faudra dès lors vérifier que cette déformation n'est pas utilisée pour créer des cycles de tailles différentes de celle pour laquelle le fragment de cycle a été conçu :

où un cycle à six est construit à partir d'une sous-structure destinée à former un cycle à cinq. Il faudrait donc interdire ce genre de chose pendant le dessin.

2. Proposition d'une solution

2.1. Mode de description

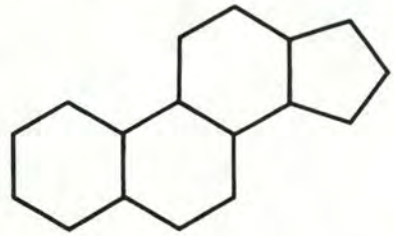
Essentiellement à cause des cycles, il semble difficile, voire impossible d'obtenir un dessin composé de sous-structures indissociables, introduites telles quelles et reliées les unes aux autres par l'intermédiaire de leurs prolongements. Le même problème se présente bien entendu pour la description directement déduite du dessin. Cela, bien entendu, si l'on considère des sous-structures plus complexes qu'un atome seul ! En fait, beaucoup de molécules seront composées de sous-structures qui se recouvrent partiellement. Et c'est ce recouvrement qu'il n'est pas aisé de décrire. Il est cependant intéressant de conserver une description d'une molécule via un ensemble de sous-structures la composant puisqu'elle est proche du point de vue du chimiste. D'où l'idée de procéder à une description selon deux points de vue :

1) Tout d'abord, obtenir une description de la molécule dans laquelle tous les atomes seraient considérés comme identiques. On obtient ainsi les

relations entre les cycles et les chaînes composant le squelette de la molécule.

2) Il reste ensuite à décrire les fonctions portées par la molécule en considérant cette fois les cycles comme n'étant plus que des chaînes d'atomes.

La description du squelette est certainement intéressante. Par exemple, celui présenté ci-dessous se retrouve dans les différents stéroïdes qui jouent un rôle important dans le monde des organismes vivants [5].



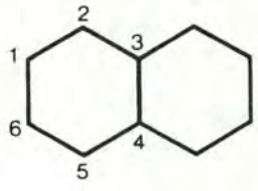
Bien qu'ils diffèrent par leurs fonctions ou leurs activités, tous les stéroïdes dérivent d'une même molécule et sont regroupés dans une même classe caractérisée par le motif tétracyclique précédent.

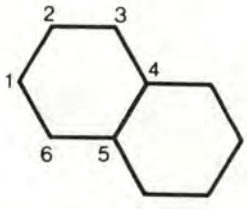
Quant à la description des fonctions, il va sans dire qu'elle est obligatoire puisque c'est par elles que passe en général la réactivité d'une molécule.

2.2. Définition des sous-structures

En fonction de ce que nous avons dit, il apparaît évident qu'à partir de maintenant, il va probablement être nécessaire de différencier les sous-structures utilisées dans la *description* de la molécule de celles utilisées pour la *construction* de la molécule.

Plusieurs solutions se présentent à nous. La plus simple est l'emploi d'une seule sous-structure pour construire tout le squelette de la molécule : une simple liaison que l'on pourrait déplacer et orienter de façon quelconque. Il serait ainsi possible de construire n'importe quel motif, comme cycles et chaînes, les hétéroatomes et liaisons multiples étant ajoutés par la suite. Mais cette solution est vraiment rudimentaire et très peu agréable pour la construction des cycles. Si la difficulté de dessin vient des cycles, pourquoi ne proposerait-on pas de tels motifs prédéfinis ? C'est la solution que nous avons finalement retenue. Nous tenterons de résoudre le problème lié à la nécessité d'avoir plusieurs sous-structures pour chaque taille de cycle. Nous verrons qu'il faudra accepter un certain recouvrement des sous-structures pour cela. Les liaisons constituant les chaînes de la molécule continueront à être introduites liaison par liaison, ce qui semble constituer une solution raisonnable tant du point de vue de la réalisation et que de celui de l'utilisation. Il faut encore préciser que d'une façon ou d'une autre, les sommets des cycles devront être numérotés. C'est inévitable pour obtenir les relations de positions entre les différents éléments de la molécule. Ces sommets étant différenciés, il y aura plusieurs possibilités de construction d'un même produit :





etc...

L'étape de description devra donc rendre cette numérotation relative plutôt qu'absolue.

Il reste maintenant à introduire les fonctions. Il aurait peut-être été possible d'employer des sous-structures

prédéfinies. Mais alors, que de problèmes pour amener la fonction à recouvrir une partie du squelette déjà créé. En effet, une sous-structure représentant une fonction serait forcément proposée selon une géométrie prédéterminée. Celle-ci devra dès lors pouvoir être "adaptée" pour épouser la forme de la partie du squelette à recouvrir. De plus, bon nombre de problèmes évoqués précédemment à propos de ces fonctions réapparaissent : composition de fonctions, taille du dictionnaire de fonctions. Nous avons donc préféré, à cette possibilité, l'introduction un par un, sur le squelette, des hétéroatomes et liaisons multiples constitutifs des fonctions. Cette solution nécessitera bien entendu, au moment de la phase de description des molécules, une reconnaissance de ces fonctions.

Remarque : Elle n'est pas réalisable suite aux problèmes déjà évoqués, mais une autre solution était envisageable. Les fonctions auraient été introduites sous forme de sous-structures complètes, contrairement, cette fois, aux cycles. Et une phase de reconnaissance des cycles et des chaînes aurait été, ici, nécessaire puisque ces cycles et ces chaînes seraient constitués de plusieurs sous-structures. Parmi celles-ci, nous aurions, évidemment, la liaison simple qui doit être maintenue, une molécule n'étant pas composée uniquement de fonctions chimiques.

En conclusion, nous pouvons donc dire qu'en ce qui conserne le squelette de la molécule et essentiellement les cycles, les sous-structures utilisées pour la construction et la description sont sensiblement les mêmes. On imagine facilement aussi que la base de la description des chaînes sera la liaison simple. Il en va par contre tout autrement pour les fonctions. Celles-ci seront en effet construites par modifications du squelette (caractérisation de certains atomes par un symbôle chimique ou modifications de liaisons simples en liaisons multiples). Ce sera donc au moment de la description qu'il faudra "recomposer" les sous-structures utiles.

CHAPITRE II
REALISATION D'UNE INTERFACE GRAPHIQUE POUR L'INTRODUCTION DE MOLECULES

1. Introduction

Comme nous l'avons laissé entendre, l'utilité d'une base de données en chimie organique ne sera réelle que si l'introduction des réactions et donc des molécules ne devient pas une tâche fastidieuse pour le chimiste. Idéalement, le dessin d'une molécule ne devrait pas prendre plus de temps sur un écran que sur une feuille de papier. Il est donc important que l'interface graphique soit simple à utiliser tout en offrant au chimiste des outils suffisamment puissants pour la construction aisée et complète d'une molécule ainsi que des motifs un peu plus élaborés qu'une simple liaison.

Nous allons donc essayer de proposer quelques idées au sujet de ces outils et de ces motifs. La réalisation pratique va certainement poser quelques problèmes parmi lesquels nous allons tenter de mettre les principaux en évidence. Nous allons ensuite décrire une solution réalisée sur un micro-ordinateur. Celle-ci ne sera probablement pas parfaite mais aura l'avantage de présenter la simplicité avec la laquelle des molécules peuvent être introduites dans un système informatique.

2. Moyens d'introduction

Il est facile d'imaginer la nécessité d'un moyen capable de déplacer facilement les sous-structures pour construire une molécule. Nous avons choisi de faire appel à un outil très pratique : une souris. Comme nous le verrons, celle-ci nous permettra de positionner correctement les différents motifs nécessaires au dessin d'une molécule. Les déplacements de la souris seront interprétés, selon la situation, comme des translations ou des rotations des motifs. Grâce à celle-ci, il sera également très facile de réaliser un choix dans un menu simplement en "cliquant" l'option souhaitée. La souris semble donc être un choix idéal pour notre réalisation.

Celle-ci sera réalisée sur un compatible IBM PC avec une résolution graphique de 320 sur 200 points qui, bien que faible, sera quand-même suffisante à la construction claire d'une molécule.

3. Fonctions offertes

Il sera possible de choisir le motif à introduire dans la molécule grâce à un menu. Le choix sera réalisé à l'aide de la souris. Quatre types de sous-structures seront proposés : des cycles, des liaisons simples, des liaisons multiples et enfin des symboles chimiques. Dans le cas des cycles, il faudra bien sûr en préciser la taille (par exemple, choix possibles de 3 à 8 atomes).

Une fois un motif choisi, il sera possible de l'ajouter à la molécule en construction par "accrochage" aux motifs

existants ou de supprimer une partie de cette molécule correspondant à ce motif, simplement par "superposition". Nous entendons par superposition, le recouvrement, point par point, d'un motif existant dans la molécule couramment dessinée à l'aide du motif choisi.

Il faudra donc pouvoir déplacer l'élément choisi sur l'écran puis, l'emplacement désiré atteint, lui donner l'orientation adéquate. Il faut bien sûr avoir la possibilité, à tout moment, de retourner à la sélection des motifs.

En plus de la sélection des motifs, il serait intéressant que le menu offre une option de translation de la molécule sur l'écran. Il serait en effet assez frustrant pour l'utilisateur de devoir recommencer tout son dessin à cause d'un mauvais positionnement initial empêchant toute possibilité d'extention sur une extrémité de la molécule.

Moyennant quelques aménagements que nous allons décrire immédiatement, ce schéma devrait permettre l'introduction rapide de molécules.

4. Aménagements

Les principes de "superposition" et "accrochage" sont bien beaux, mais cela ne doit pas devenir fatiguant pour l'utilisateur. Il est en effet hors de question que celui-ci prenne le temps et la patience nécessaires à la superposition point par point de deux motifs. Ce sera donc au système de prendre cela en charge en considérant que deux points suffisammant proches sont identiques. Il restera donc à l'utilisateur à amener, de façon approximative, le motif à

l'endroit désiré tout en lui donnant une bonne orientation. Cette opération sera réalisée en deux étapes bien distinctes, à savoir, une translation suivie d'une rotation autour d'un point de la molécule qui aura été fixé de façon définitive après la translation.

5. Réalisation

5.1. Définitions

Il est d'abord nécessaire de définir différents termes qui vont être utilisés dans le texte qui va suivre. En effet, pour des raisons de facilité, certains de ceux-ci vont différer quelque peu de leur sens chimique. Il nous faudra tout d'abord différencier les termes de *liaison simple* et de *liaison de cycle*. Ce dernier référencie un segment reliant deux sommets d'un cycle tandis que le premier se rapporte à un segment reliant deux sommets dont la seule restriction est qu'ils n'appartiennent pas au même cycle. Un *sommet* est une extrémité d'une liaison (simple ou de cycle). En chimie organique, ces sommets, s'ils ne sont caractérisés par aucun symbole chimique, sont considérés comme étant des atomes de carbone. Ici, nous parlerons de sommets portant un *atome* uniquement s'ils sont carctérisés par un symbole chimique, bien que normalement, chaque sommet soit un atome. Enfin, nous dirons qu'une liaison simple ou de cycle comporte une *liaison multiple* si les deux sommets de la liaison sont reliés par une liaison multiple au sens chimique du terme.

5.2. Problèmes généraux

Il ne faut pas oublier que le dessin de la molécule est destiné par la suite à être traité. Cependant, si l'on ne

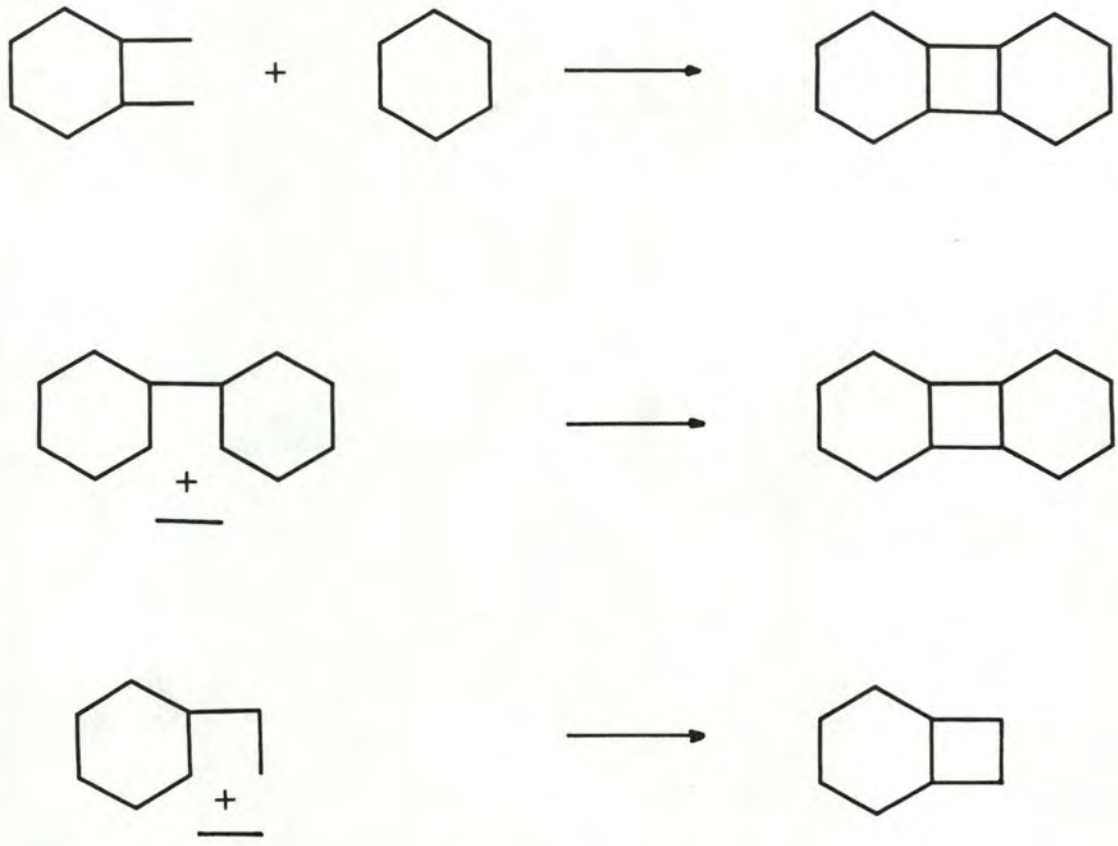
prend pas quelques précautions, la molécule pourra être différente du dessin. En effet, les possibilités de dessin offertes à l'utilisateur lui permettront aisément de superposer deux cycles de même taille, de superposer une liaison simple sur une liaison d'un cycle ou de construire un cycle sur une liaison simple préexistante. Cela introduit une redondance évidente qui risque de poser des problèmes lors du traitement de la molécule. Cette redondance doit donc être éliminée pendant la phase de dessin. Il suffit pour cela d'appliquer une règle très simple :

> Une liaison simple sera refusée si elle est superposée à une liaison de cycle ou à une autre liaison simple. De même, un cycle sera refusé si une de ses liaisons est commune à une liaison simple ou si toutes ses liaisons sont communes aux liaisons d'un autre cycle de même taille (en fait, le cas de deux cycles superposés).

Il est à noter que cette règle n'interdit pas la mise en commun d'une liaison de deux cycles différents. Cela risquerait autrement de poser quelques problèmes de construction!

Outre ce problème de superposition de motifs, qui est finalement facilement identifiable, il sera important d'imposer le plus possible à l'utilisateur l'emploi des sous-structures disponibles pour la construction des cycles. Autrement dit, il faudrait rendre difficile la construction de cycles par combinaisons de plusieurs sous-structures et ainsi, éviter la formation de cycles "parasites" non détectés.

Exemples :



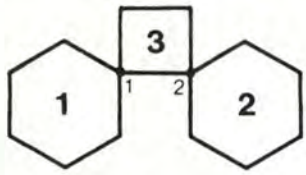
Ces cas de constructions devraient être interdits.

Nous allons pour cela tenter l'introduction de nouvelles règles dans le sytème de dessin. Celles-ci sont décrites dans les lignes qui suivent. Il nous faut cependant faire quelques hypothèses préalables quant à l'introduction des cycles. Nous avons en effet vu plus haut que ceux-ci seront introduits par une translation suivie d'une rotation. La première se fera par déplacement d'un des sommets du cycle, les autres suivant ce déplacement. Une fois ce sommet positionné définitivement, l'autre extrémité d'une des liaisons auxquelles appartient le sommet maintenant fixé pourra décrire un mouvement de rotation autour de ce point d'attache en entraînant bien sûr avec lui tout le cycle. Une fois ce deuxième sommet positionné, le cycle sera définitivement fixé.

1ère règle :

Si le premier sommet fixé d'un cycle est commun à un sommet et que ce sommet appartient à un autre cycle, le deuxième sommet ne pourra être commun qu'à un autre sommet de cet autre cycle (et les deux sommets appartiennent bien sûr à une même liaison de cet autre cycle) ou à aucun autre.

Ce qui signifie que le dessin suivant est impossible à obtenir s'il est fait dans l'ordre des numéros de cycle.



La règle interdit en effet au sommet 2 du troisième cycle d'être commun à un sommet du deuxième. On évite ainsi la formation possible d'un cycle supplémentaire dans le cas où les cycles 1 et 2 auraient déjà été reliés d'une façon quelconque.

2ème règle :

Si le premier sommet d'un cycle est commun à un sommet d'une liaison simple, le 2è sommet du cycle ne pourra être commun à aucun autre sommet.

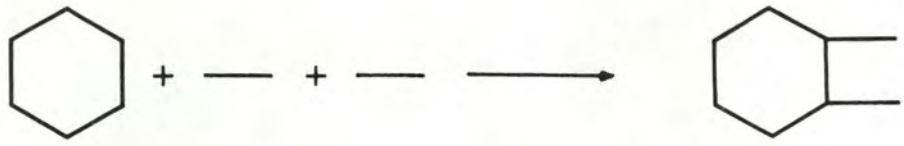
3ème règle :

De plus, à sa construction, un cycle ne peut avoir aucun de ses sommets commun avec un sommet d'une liaison simple à l'exception du premier sommet fixé ou que le sommet de la liaison simple soit déjà commun au sommet d'un autre cycle.

Ces deux dernières règles interdisent les constructions du style de la figure suivante qui peut présenter des risques si elles sont réalisées dans cet ordre :

liaisons
déjà dessinées

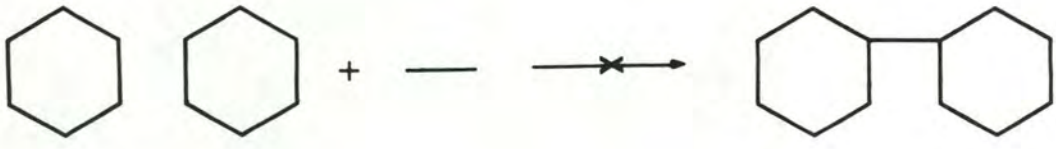
Elles n'empèchent pas de construire le même motif mais dans un ordre différent qui, lui, ne présente aucun danger.



4ème règle :

> Quand on introduit une liaison simple, seul le premier sommet peut être commun à un autre sommet. C'est-à-dire qu'aucune liaison simple ne pourra relier deux sous-structures du dessin au moment de son placement.

Il sera donc impossible de construire un cycle à l'aide de liaisons simples. Les dessins suivants, réalisés dans l'ordre indiqué seront donc interdits :



On peut noter que ces quatre dernières règles recouvrent partiellement la première décrite. Elles n'interdisent cependant pas à deux cycles de même taille de se superposer.

Il s'est avéré à l'usage qu'une nouvelle règle allait être nécessaire pour améliorer la clareté du dessin. Nous allons d'abord en donner l'énoncé et ensuite l'intérêt.

5ème règle :

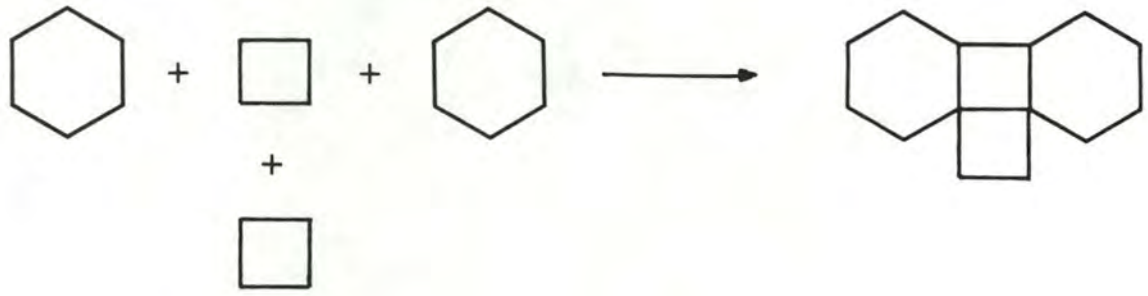
> Si le premier sommet d'un cycle n'a pas été ajusté (et donc le deuxième non plus) ou si seul ce sommet l'a été, alors, aucun de ses autres sommets ne peut être proche d'un sommet quelconque du dessin.

Quand nous parlons d'ajustage, il s'agit simplement de la superposition automatique de deux sommets suffisamment proches de deux motifs différents.

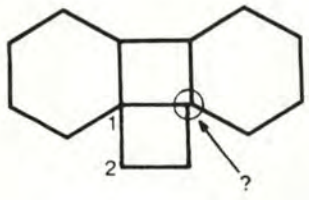
Nous allons développer les intérêts d'une telle règle. Le premier point est essentiel : dans certains cas, cette règle interdira la construction de cycles non détectés comme celui de l'exemple suivant :

En effet, jusqu'à présent, aucune règle n'empêchait l'introduction de ce cycle à quatre en procédant de la sorte (un seul des deux sommets ajusté à l'introduction).

La seule façon de construire ce dessin serait d'indiquer réellement au système l'existence d'un quatrième cycle de la manière suivante :



Outre ce dernier problème, un second intérêt de cette règle est d'empêcher la construction de certains dessins pouvant prêter à confusion pour l'utilisateur. Il faut en effet rappeler que le système n'ajustera les deux premiers sommets d'un nouveau cycle que sur un même cycle (règle N° 1). Reprenons pour cela l'exemple précédent et voyons ce qu'il peut se passer.

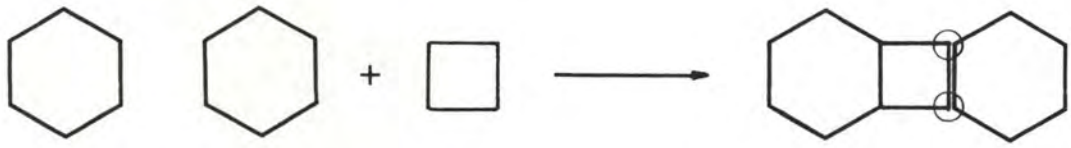


Le système ne garantit pas à l'utilisateur que le sommet entouré du cyclobutane se superposera exactement à celui du second cycle à 6 carbones, bien qu'il puisse en avoir l'apparence s'il en est très proche.

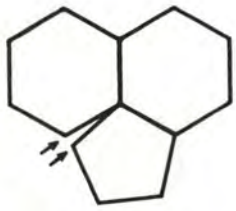
Ce problème ne se serait pas posé si le deuxième cycle à 4 carbones ainsi formé avait ètè introduit explicitement, ce qu'impose la 5ème règle. Toutes les ambiguïtés de ce type ne peuvent cependant être levées sous peine de réduire alors

les possibilités de dessins des molécules.

Ex. :



ne peut être interdit sans rendre à la fois impossible le dessin suivant où les atomes marqués d'une flèche seraient trop proches l'un de l'autre :



6. Réalisation pratique

Nous allons aborder, ici, de façon plus concrète la méthode de construction d'une molécule ainsi que les problèmes qui y sont liés.

6.1. Méthode de construction

Nous avons déjà vu que la construction d'une molécule allait essentiellement comporter quatre phases :

- introduction de cycles
- introduction de liaisons simples
- introduction de liaisons multiples
- introduction de symboles d'atomes

Les deux premières phases devront respecter les règles précédemment définies. Quant aux deux dernières, il va de soi qu'une liaison multiple ne pourra être introduite que si

la liaison simple ou de cycle correspondante existe. Un atome ne pourra, lui, être introduit que si le sommet correspondant existe.

Comme nous l'avons déjà dit, l'introduction se fera au moyen d'une souris. A aucun moment du dessin il ne sera nécessaire d'employer le clavier. Le bouton gauche de la souris servira généralement pour afficher une nouvelle sous-structure, la touche droite étant utilisée pour l'effacement. Un appui simultané sur les deux boutons pemettra de revenir au menu général.

6.1.1. Introduction de cycles

Il n'y a plus grand chose à dire sur ce point vu que la méthode d'introduction a déjà dû être décrite précédemment. L'appui sur un bouton de la souris fera apparaître le cycle séléctionné à la position du curseur. Tant que ce bouton sera maintenu enfoncé, le motif pourra être déplacé sur l'écran. Le sommet du cycle qui était représenté par un point plus important durant le déplacement du cycle sera fixé à l'endroit qu'il occupait lorsque le bouton de la souris a été relaché. Cependant, si un autre sommet était suffisamment proche de ce point, le sommet "marqué" du cycle sera automatiquement déplacé vers ce point (suivi par le cycle complet). C'est maintenant le deuxième sommet du cycle qui est marqué par le point plus important. Celui-ci peut alors subir un mouvement de rotation autour du premier point tant qu'aucun bouton de la souris n'est de nouveau enfoncé. Ce deuxième sommet sera alors fixé dans les mêmes conditions que le premier avec rotation éventuelle vers le sommet le plus proche. Si c'est le bouton gauche qui est pressé, le nouveau cycle sera ajouté à la molécule si sa position répond aux règles définies, sinon, il sera effacé. Si le

bouton droit a été enfoncé et que le nouveau cycle en recouvrait un autre de même taille, ce dernier sera supprimé de la molécule et le nouveau cycle est bien sûr effacé. Dans le cas contraire, le nouveau cycle est simplement effacé.

6.1.2. Introduction de liaisons simples :

Les manipulations des liaisons simples sont tout à fait semblables à celles faites sur les cycles. Et de nouveau, l'introduction d'une nouvelle liaison simple ne sera acceptée que si elle répond aux règles la concernant.

6.1.3. Introduction de liaisons multiples :

Il existe en chimie organique des liaisons doubles ou triples représentées respectivement par deux ou trois traits (dans ce paragraphe, les dénominations des liaisons répondront à leur sens chimique). Ces deux motifs seront obtenus par les mêmes manipulations. A l'aide de la souris, le curseur sera placé près d'un des sommets devant participer à une liaison multiple. Il faudra alors presser un bouton quelconque de la souris et le maintenir enfoncé jusqu'à ce que le curseur soit proche du deuxième sommet qui devra participer à la liaison multiple. Si la liaison de départ était simple, elle devient double, si elle était double, elle devient triple et, finalement, si elle était triple, c'est la liaison simple qui est reformée. La même opération permet donc aussi bien la formation d'une liaison multiple que sa suppression sur base d'une liaison simple.

6.1.4. Introduction de symboles

Après avoir choisi cette option du menu, un tableau de Mendeleiev, contenant tous les atomes, sera représenté à l'écran. Il suffira de sélectionner le symbole désiré avec la souris. Celui-ci pourra alors être déplacé sur le dessin de la molécule, tout bouton de la souris relaché. Une fois suffisammant proche du sommet désiré, il suffira de presser le bouton gauche de la souris pour introduire le symbole. S'il en existait déjà un, il est simplement remplacé par le nouveau. Si le bouton droit avait été enfoncé, le symbole éventuellement présent sur le sommet concerné aurait été effacé quel qu'il ait été.

Les caractères composant le symbole seront centrés sur le sommet choisi.

Il nous reste maintenant à parler de la possibilité de déplacer toute la molécule sur l'écran. Elle n'intervient absolument pas dans le dessin même de la molécule mais elle pourra être bien utile dans certains cas.

6.1.5. Translation de la molécule :

Après sélection de ce choix, il suffit de presser un bouton de la souris pour que molécule soit entourée par un rectangle la symbolisant. Celui-ci peut alors être déplacé n'importe où sur l'écran. Une fois le rectangle bien positionné, il suffit d'enfoncer le bouton gauche de la souris pour déplacer réellement la molécule. Une pression sur le bouton droit annule le déplacement.

6.2. Problèmes du dessin des motifs

Du point de vue même du dessin, seuls les cycles et les symboles chimiques semblent présenter quelques difficultés. En effet, pour être compatible avec la taille du dessin, les caractères représentant les symboles devront être codés dans une matrice de trois points en largeur et cinq en hauteur. Deux caractères seront séparés par un point. Quant aux cycles, il serait intéressant de trouver une méthode générale permettant de les tracer avec une taille variable (restant cependant compatible avec la définition de l'écran. C'est en regard de cette définition que nous avons décidé de limiter la taille à huit atomes).

Il reste enfin à trouver une méthode permettant de déplacer un motif de façon quelconque. Il faudrait en effet qu'il puisse voyager aussi bien sur une partie vierge de l'écran que sur une autre partie occupée par le dessin en cours et cela, bien évidemment, sans modifier ce dessin.

6.3. Problèmes liés à la suppression de motifs

Les suppressions ne peuvent être réalisées de façon simple par le système. En effet, si en ce qui concerne la mémorisation interne de la molécule, il suffit de supprimer les sommets en rapport avec le motif concerné (et encore, sans compter les motifs en rapport avec celui qui est supprimé. Nous aborderons ces problèmes), il n'en va pas de même avec le dessin. Nous allons donc examiner les principaux cas possibles et voir ce qu'il en est aux niveaux de la mémorisation des sommets et motifs et de leurs représentations sur l'écran.

6.3.1. Suppression d'un cycle :

Sur l'écran, l'effacement d'un cycle consiste à effacer toutes les liaisons qui le composent. Cependant, si celui-ci était accolé à un ou plusieurs autres cycles, les liaisons qu'il avait en commun avec ceux-ci ne doivent pas être effacées. En mémoire, il suffit de supprimer le cycle.

6.3.2. Suppression d'une liaison simple :

Cela ne devrait guère présenter de problème. Si la liaison effacée avait des sommets communs à d'autres, les points correspondant ont été effacés. Bien que ce soit peu visible, ils pourront éventuellement être restaurés.

6.3.3. Problèmes supplémentaires :

Ceux-ci sont dus aux liaisons multiples et aux symboles chimiques qui viennent se greffer sur les sommets et liaisons existantes.

i) liaisons multiples

Si une liaison simple est supprimée et qu'elle possédait une liaison multiple, celle-ci doit bien entendu être supprimée également. Il en va de même pour chaque liaison de cycle supprimé sauf si les liaisons concernées ne sont pas supprimées parce qu'elles appartiennent aussi à un autre cycle.

ii) symboles chimiques

Si on supprime une liaison simple ou de cycle et qu'un des sommets possédait un symbole chimique, celui-ci devra être supprimé s'il n'est commun à aucune autre liaison. Mais les problèmes posés vont essentiellement être d'ordre graphique. Il faut en effet rappeler qu'un symbole est

centré sur le sommet qu'il occupe, c'est-à-dire qu'il a effacé partiellement les liaisons qui avaient ce sommet comme extrémité. La suppression d'un symbole nécessitera donc la restauration de ces liaisons, sans oublier éventuellement celle des liaisons multiples. Il faudra encore faire attention à l'effacement d'une liaison dont un sommet (ou deux) est commun à un symbole qu'il ne faut pas effacer. Sinon, on risque d'effacer partiellement les caractères composant ce symbole.

6.4. Problèmes liés à l'addition de motifs

Si, à première vue, la suppression de motifs semblait présenter quelques problèmes, l'addition va également en poser, essentiellement à cause de la présence de symboles chimiques. En effet, rien n'interdit l'addition d'un motif sur un sommet possédant déjà un symbole chimique. Il faudra donc veiller à ne pas surimprimer ce symbole.

Nous venons de décrire les problèmes principaux. On pourrait encore en trouver d'autres dus, par exemple, à la faible définition de l'écran. En effet, si deux liaisons ont un sommet commun et que les deux autres sommets sont très proches, la suppression d'une liaison risque d'effacer partiellement l'autre. Nous n'avons pas pris ces problèmes en compte. Il faut cependant préciser que la molécule est chaque fois redessinée entièrement après avoir effectué un choix dans les menus. Les défauts non corrigés qui pourraient alors éventuellement apparaître seront donc automatiquement supprimés avec l'effacement du dessin à la suite des retours aux menus.

7. Solution proposée

D'après ce qui vient d'être dit, on se rend vite compte que la vitesse d'exécution va être quelque chose de déterminant. En effet, le nombre de tests à réaliser va être important et directement lié à la taille de la molécule en cours d'introduction. Nous avons donc choisi de réaliser l'implémantation en langage C [6]. Les routines de gestion du graphisme et de gestion de la souris seront cependant écrites en assembleur 8086 [7]. Nous allons commencer par décrire le rôle de ces dernières.

7.1. Gestion du graphisme et de la souris

Ces procédures seront facilement accessibles via un programme en C et joueront un rôle d'interface entre les caractéristiques de la machine et le programme de dessin de la molécule proprement dit. De plus, elles apporteront déjà certaines solutions aux problèmes qui ont été posés.

7.1.1. Gestion du graphisme

LINE permettra de tracer une ligne entre deux points. Leurs coordonnées doivent se trouver dans les limites de l'écran sinon, la droite ne sera pas dessinée. Cette ligne pourra avoir les caractéristiques suivantes :

- tracé avec mémorisation des points recouverts
- perte de mémorisation des points recouverts
- effacement avec suppression ou non des points recouverts

Il y a cependant une condition : un point d'intersection ne peut mémoriser qu'un recouvrement à la fois. Donc, si un point est surimprimé deux fois, la

suppression des deux derniers points introduits effacera inévitablement le premier aussi. Dans certains cas, ce problème pourra être évité en utilisant une autre caractéristique de la droite tracée :

- le point de départ de la droite peut ne pas être affiché.

En effet, dans notre réalisation, le seul cas où plus de deux sommets pourraient être superposés est la création des cycles, chaque sommet étant l'intersection de deux droites. Si le premier point de ces dernières n'est jamais affiché et si elles sont tracées dans un ordre précis, il n'y aura plus de superposition au niveau des sommets. On obtient donc un motif que l'on pourra déplacer sans problème.

Une dernière caractéristique importante de cette procédure est qu'une même droite sera toujours tracée sur les mêmes points même si les extrémités de la droite sont introduites dans un ordre différent.

HLINE permettra de tracer très rapidement des lignes horizontales. Cette routine ne possède aucun autre paramètre que les points initial et terminal de la droite. Elle sera utilisée pour le tracé rapide de tableaux.

VLINE sera semblable à HLINE mais pour le tracé de droites verticales et sera aussi utilisée essentiellement pour le tracé rapide de tableaux.

PRCARGR dessinera une lettre dans une matrice 3 x 5 centrée sur les coordonnées introduites. Si la lettre ne peut être représentée entièrement (coordonnées trop près du bord), elle ne sera pas dessinée.

7.1.2. Gestion de la souris :

WINDCURS permettra de définir une fenêtre dans laquelle le curseur pourra se déplacer.

GETPOSC nous fournira la position du curseur ainsi que l'état des boutons de la souris.

PUTPOSC permettra de fixer le curseur de la souris à un endroit déterminé de l'écran.

SELCURS permettra de choisir une forme de curseur par son numéro identifiant. On disposera d'un curseur en forme de croix, d'un autre en forme de point et finalement d'une flèche.

FORMACUR permettra de former un curseur alphabétique composé de deux lettres au maximum. Cela sera bien utile pour le positionnement des symboles chimiques.

7.2. Gestion du dessin d'une molécule

Comme nous l'avons vu plus haut, il faut envisager ici deux points de vue importants : celui de la mémorisation de la molécule et celui de son dessin.

7.2.1. Mémorisation de la molécule

Les sommets seront répartis dans quatre listes correspondant chacune à l'une des phases de construction de la molécule.

i) liste de cycles

Pour chaque cycle, cette liste contiendra les informations successives suivantes :

- la taille du cycle,
- pour chaque sommet du cycle, sa coordonnée X suivie de la coordonnée Y.

ii) liste de liaisons simples

Pour chaque liaison simple, cette liste contiendra les coordonnées des deux sommets de la liaison, toujours dans l'ordre X,Y.

iii) liste des liaisons multiples

Pour chacune d'elles, la liste contiendra les informations suivantes :

- multiplicité de la liasion (2 ou 3),
- les coordonnées des sommets impliqués dans cette liaison multiple.

iv) liste des symboles d'atomes

Pour chaque symbole introduit, cette liste comportera les informations suivantes :

- le code des deux lettres caractérisant le symbole. Si une seule lettre est nécessaire, le deuxième code prendra une valeur spéciale.
- les coordonnées du sommet portant ce symbole.

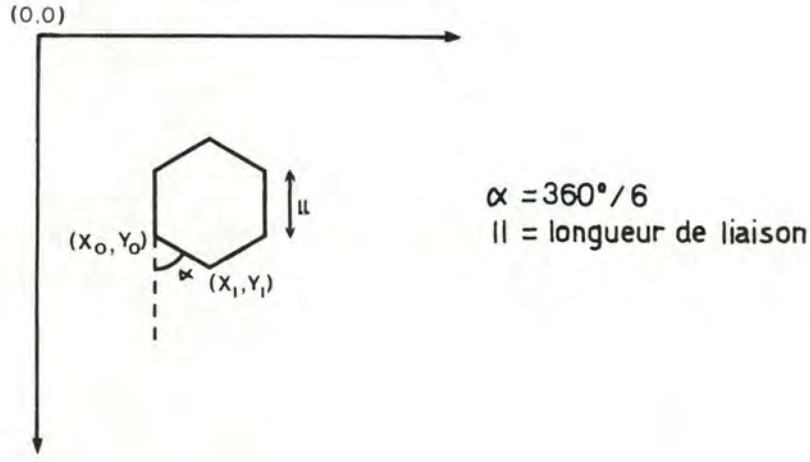
7.2.2. Dessin de la molécule

Nous allons ici décrire rapidement chacune des quatre phases du dessin d'une molécule. Pour obtenir plus de détails, il suffira de consulter le listing du programme. Chaque procédure est précédée d'un commentaire pour en rappeler les fonctionnalités.

7.2.2.1. Introduction des cycles

Nous avons donc réalisé une procédure capable de tracer des cycles de taille quelconque, cette méthode semblant la plus intéressante. Nous allons décrire la méthode utilisée et essayer de mettre en évidence les problèmes posés par la réalisation pratique.

Nous allons prendre comme exemple le tracé d'un cycle à six atomes.

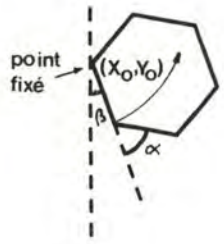


De manière générale, *alpha* = 360^{o} / nombre de sommets. On peut facilement obtenir les coordonnées d'un sommet connaissant celles du précédent. Nous allons décider d'un sens de parcours anti horlogique des sommets. Ce sens sera tout naturellement fixé par le système de coordonnées écran, le point (0,0) étant le coin supérieur gauche de l'écran.

Le sommet (Xi, Yi) sera donc obtenu de la façon suivante :

$$X_i = ll \times \cos(alpha \times i) + X_{i-1}$$
$$Y_i = ll \times \sin(alpha \times i) + Y_{i-1}$$
avec $i > 0$

A partir de là, on voit qu'il est très facile de faire subir au cycle un mouvement de rotation autour de l'un de ses sommets.



Les coordonnées des sommets seront alors obtenues de la façon suivante :

$$X_i = ll \times \cos(alpha \times (i-1) + keta) + X_{i-1}$$
$$Y_i = ll \times \sin(alpha \times (i-1) + keta) + Y_{i-1}$$

avec $i > 0$

Quant à la translation, on aura tout intérêt à travailler en coordonnées relatives à un des sommets puisque le cycle ne change jamais d'orientation pendant cette étape.

D'un point de vue théorique, cela fonctionne très bien. En pratique, des problèmes vont se poser à cause des erreurs d'arrondi. En effet, pour des raisons de performances, tous les calculs vont se faire sur des entiers qui sont d'ailleurs tout à fait adaptés aux coordonnées écran. Cela signifie donc qu'il est hors de question de commencer à calculer des sinus et des cosinus pendant le dessin. Ces valeurs seront donc calculées avant l'exécution proprement dite et multipliées par la longueur de liaison (ll) pour pouvoir être mémorisées sous forme d'entiers. Cependant, même si ce problème n'existait pas, il y en aurait toujours un pour la recherche de l'angle *keta*. En effet, celui-ci ne peut être obtenu qu'à partir de coordonnées écran et

notamment de la position du curseur manipulé par la souris. Voyons d'abord comment obtenir cet angle. Si dans le dessin précédent, le curseur est directement lié au sommet du cycle de coordonnées (X1,Y1), *beta* sera l'arc tangente de la pente de la droite passant par le curseur et (X0,Y0). Nous verrons comment cet angle est obtenu pratiquement. Regardons maintenant l'exemple suivant où deux cycles de taille et de position identiques ont été obtenus à partir d'un point origine différent.

Il est peu probable que, dans tous les cas d'orientation possibles, l'angle *beta 2* soit identique à l'angle *gamma 1*. Ce dernier est effectivement obtenu par la somme de *beta 1* et *alpha*. Or, celui-ci est un angle calculé de façon précise tandis que *beta 1* et *beta 2* ne peuvent prendre qu'un nombre très limité de valeurs, cela étant dû à la résolution de l'écran. On va donc déjà obtenir une erreur sur les valeurs de sinus et cosinus. De plus,cette erreur va être multipliée par la longueur de liaison. Lorsque ces résultats vont finalement être approchés aux entiers les plus proches, on risque d'avoir un décalage de un point entre deux sommets théoriquement identiques. Et on constatera bien par quelques essais que pour certaines tailles et orientations de cycle, le recouvrement de deux cycles n'est pas parfait. Cela peut sembler peu important pour le dessin en lui-même. Mais quand on se rappelle que les cycles sont identifiés par les coordonnées de leurs sommets, cela devient préoccupant. En effet, l'effacement d'un cycle risque d'être une opération

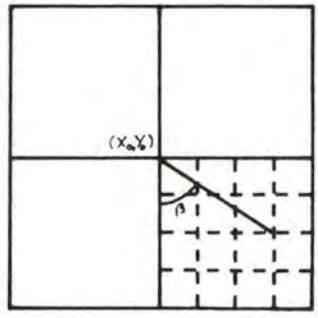
présentant pour le moins quelques chances d'insuccès si l'utilisateur ne se rappelle plus par quel sommet il l'avait commencé!

Voyons maintenant comment on peut règler le problème de la superposition précise de deux cycles. Nous avons vu qu'après translation et rotation, deux sommets du cycle sont fixés définitivement et peuvent éventuellement être communs à un autre cycle. Si c'est le cas, il n'est pas impossible que cet autre cycle soit de même taille et à la même position (et même orientation) que le nouveau. Pour vérifier si l'utilisateur a essayé de faire une superposition, il suffira donc de parcourir la liste des cycles en regardant s'il en existe un de même taille, ayant deux sommets de coordonnées tout à fait identiques (ceux fixés par translation et rotation) et dont les autres soient suffisamment proches que pour être considérés comme identiques (Cela est réalisé par la procédure ADJCYCLE). Dans ce cas, les deux cycles pourront être identifiés.

Le premier sommet a été fixé, lui, après translation du cycle, de façon semblable par la procédure PROCH1SOM tandis que le deuxième l'a été par la procédure PROCH2SOM. Ces deux procédures et leurs conditions d'appel répondent aux règles énoncées précédemment quant à la mise en commun de sommets de cycles.

Revenons enfin au calcul de la pente. Comme on l'a vu, on peut se permettre de ne retenir que des angles entiers. (Les calculs de sinus et de cosinus ne se feront également que sur des angles entiers et l'angle alpha sera également arrondi à l'angle entier le plus proche.) Considérons maintenant le sommet (X1,Y1) qui est à la place occupée par le curseur pendant la rotation. Celui-ci parcourt normalement un cercle centré sur le sommet (X0,Y0) et de

rayon ll (longueur de liaison). Nous allons donc tout d'abord fixer une fenêtre de déplacement du curseur qui sera un carré centré sur (X0,Y0) et de côté llx2+1. Ne considérons maintenant plus qu'un quart de ce carré. Nous définissons ainsi une matrice de points de taille (ll+1)x(ll+1). Et comme on le voit très bien sur la figure suivante, à chaque point de cette matrice pourra correspondre un angle *beta* calculable avant le dessin. Chaque valeur d'angle peut être rangée dans un tableau à deux dimensions. Et les points d'entrée dans ce tableau sont les coordonnées (relatives, bien sûr) des points de la matrice.



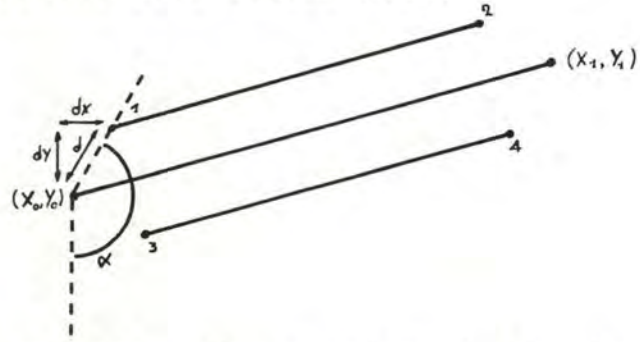
Il est bien entendu que, si chaque point du carré centré sur le sommet (X0,Y0) est accessible par le curseur via la souris, tous ces points ne sont pas pour autant des coordonnées possibles pour le sommet (X1,Y1). Celui-ci doit en effet se trouver à une certaine distance (ll) du sommet (X0,Y0). La position du curseur devra donc être recalculée et modifiée en conséquence après chacun de ses déplacements.

7.2.2.2. Introduction de liaisons simples

Les principes de translation et rotation sont tout à fait semblables à ceux utilisés dans le cas de l'introduction de cycles. Bien entendu, une procédure similaire à ADJCYCLE n'est pas nécessaire ici.

7.2.2.3. Introduction de liaisons multiples

La méthode d'introduction est ici totalement différente. Le curseur ne sert en fait qu'à sélectionner deux sommets existants. Si une liaison existe entre ces deux sommets, elle pourra être modifiée. Voyons comment la position des traits représentant la liaison multiple est obtenue. La figure suivante montre clairement comment des coordonnées peuvent être déterminées.



Un premier point est obtenu grâce à l'angle *alpha* et la distance d. Les autres points sont alors calculés de la façon suivante :

point 2 (X1-dy, Y1-dx)
point 3 (X0+dy, Y0+dx)
point 4 (X1-dx, Y1+dy)

Les coordonnées des points sont donc directement connues et il n'y a besoin ni de translation ni de rotation.

7.2.2.4. Introduction de symboles chimiques

L'introduction de symboles chimiques est extrêmement simple car elle est grandement facilitée par les procédures en assembleur. Il y a essentiellement deux étapes. La première donne au curseur la forme du symbole chimique (une ou deux lettres). Celui-ci peut alors être amené sur n'importe quel sommet existant. Il ne reste plus alors qu'à afficher définitivement le symbole en le centrant sur le sommet choisi.

7.2.2.5. Restaurations diverses

Nous avons parlé de la nécessité de restaurer certaines parties du dessin suite à des effacements, par exemple. Les principes utilisés ne sont pas très compliqués puisqu'il suffit en fait de parcourir les quatre listes décrites pour trouver les motifs communs (donc ayant des sommets communs) à celui qui a été effacé et ensuite de les restaurer. Bien entendu, s'il faut restaurer une liaison d'un cycle, on ne va pas redessiner tout le cycle.

La principale difficulté est de repérer les endroits où cela sera nécessaire et de savoir ce qu'il faut restaurer. Nous allons détailler le problème le plus important qui est celui des symboles chimiques. Voici donc cinq situations qui peuvent se présenter :

* Suppression d'un symbole :
  - il faut rechercher toutes les liaisons simples à ce sommet et les restaurer,
  - rechercher toutes les liaisons multiples communes à ce sommet et les restaurer,
  - vérifier que le deuxième sommet des liaisons restaurées n'est pas commun à un autre symbole. Si ce n'était pas le cas, il faut aussi restaurer ce symbole.

* Suppression d'une liaison simple :
  - Il faut vérifier si les sommets libérés de cette liaison comportaient des symboles.
  - Si c'est le cas, vérifier si ces sommets participent encore à d'autres liaisons.
    + si non, effacer ces symboles (et bien sûr, les supprimer de la liste)
    + si oui, il faut restaurer ces symboles.

* Suppression d'une liaison multiple :

  - Il suffit de vérifier si les sommets portaient des symboles chimiques. Si c'est le cas, il faut les restaurer.

* Ajout d'une liaison (simple ou multiple) :

  - Les tests sont les mêmes que pour la suppression d'une liaison multiple.

* Manipulations de cycles :

  - Ce sera tout à fait similaire au cas des liaisons simples, sauf en ce qui concerne le nombre de sommets à tester.

Il reste enfin à introduire une procédure de restauration complète de la molécule. En effet, chaque apparition du menu l'efface totalement. Il faudra donc la redessiner entièrement à partir des listes de sommets. Cela est en fait très simple pourvu que l'on réaffiche les symboles chimiques après tous les autres motifs.

7.2.2.6. Translation du dessin

Nous avons déjà parlé de l'utilité d'une telle possibilité. Il faut encore ajouter que si certains motifs sont dessinés trop près du bord de l'écran, ils seront partiellement ou même totalement escamotés. Une translation pourrait être utilisée pour les faire réapparaître.

Les sommets du rectangle qui symbolisera la molécule sont simplement obtenus en recherchant les coordonnées minimales et maximales de tous les sommets. Après le positionnement correct du rectangle par la souris, le déplacement réalisé en sera déduit. Celui-ci sera simplement

ajouté aux coordonnées de tous les sommets de la molécule. Puis l'écran sera effacé et le dessin restauré dans sa nouvelle position.

8. Conclusions

La solution que nous venons de décrire permet l'introduction relativement aisée d'une molécule. De plus, les motifs cycliques sont introduits en temps que sous-structures uniques, ce qui permettra leur repérage facile lors du codage de la molécule. Il faut de plus remarquer que chaque sommet est identifié sans ambiguïté par ses coordonnées. Celles-ci pourront aisément être remplacées par un numéro pour faciliter les traitements futurs.

Il faut cependant noter qu'un problème de dessin qui a déjà été évoqué précédemment n'a pas encore été abordé ici. Il s'agit de la construction d'un cycle ayant plus de deux sommets communs avec d'autres cycles tel l'exemple suivant :

Il apparaît que, dans le cadre de la solution proposée, ce problème n'en est plus un. En effet, ce qui a été fait pour deux sommets des cycles (les identifier aux sommets les plus proches, s'ils existent, après translation et rotation) peut certainement être fait pour les autres sommets du cycle. Et cela n'entrera pas en contradiction avec les règles énoncées précédemment, pourvu que les recherches de

proximité se fassent uniquement sur base des sommets de cycles et non de sommets appartenant uniquement à des liaisons simples. Le dessin de tels cycles déformés pourrait donc être implémenté sans trop de difficultés.

Il est peut-être bon d'évoquer ici que nous n'avons fait aucun test sur les valences des atomes de la molécule. Ce genre de test pourrait être implémenté lors de chaque introduction de symboles chimiques. Ce contrôle, qui sera malgré tout alourdi par la prise en compte la multiplicité des liaisons, ne pourra cependant se faire que sur le nombre *maximum* de liaisons auxquelles un atome peut participer, ce nombre n'étant pas toujours atteint. Le problème n'est cependant pas si simple car rien n'empêche l'utilisateur d'ajouter une nouvelle liaison sur un atome défini par un symbole chimique. Le contrôle de la valence devra donc être aussi réalisé pour toute introduction de cycle ou liaison si cette introduction met en cause un atome portant un symbole. Par ailleurs, si cette vérification peut-être faite pendant le dessin, il en sera autrement pour les atomes ne portant pas de symbole, donc des carbones. Tant que le dessin n'est pas terminé, ce que seul l'utilisateur peut décider, aucun test ne sera possible sur ces atomes puisqu'un symbole peut à tout moment en changer la valence. La solution la plus simple consisterait donc à faire les tests sur tous les atomes lorsque l'introduction de la molécule est terminée.

Il reste un problème non abordé : c'est le pontage des cycles.

Ex. :

Il est certain que les règles de dessin énoncées plus haut l'interdisent. Ce ne serait pourtant pas irréalisable si on

introduisait une option spéciale dans ce but. Le véritable problème ne se trouve pas en fait au niveau du dessin mais bien à celui de son codage, réalisé par la suite.

Nous allons terminer par une petite remarque qui peut prendre quelqu'intérêt lors de l'introduction complète d'une réaction. Il arrivera probablement souvent qu'un produit de réaction soit semblable à un réactif. Le système, tel qu'il est décrit permet de tirer profit de cette constation. Il est en effet facile de conserver une partie d'une molécule, comme le squelette par exemple, et de l'utiliser alors comme base pour la construction d'une nouvelle molécule.

CHAPITRE III : DESCRIPTION D'UNE MOLECULE

1. DESCRIPTION DU SQUELETTE DE LA MOLECULE

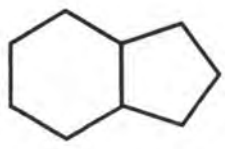
Celle-ci sera réalisée en deux étapes, à savoir la description des cycles et celle des chaînes.

1.1 Description des cycles

Comme nous l'avons vu dans l'étape de dessin de la molécule, les cycles sont introduits sous forme de sous-structures complètes. L'identification et l'isolement de ces cycles en seront donc grandement facilités. Il sera impossible de confondre les cycles avec les chaînes pour lesquelles les règles du dessin interdisent un bouclage sur elles-mêmes et donc la création de cycles sous une autre forme.

Cette description va devoir mettre en évidence les nombreuses relations pouvant exister entre les cycles. En effet, un cycle comporte plusieurs atomes dont chacun peut être relié à un autre cycle et ainsi de suite. Si une molécule ne comporte qu'un ou deux cycles liés, le problème reste simple.

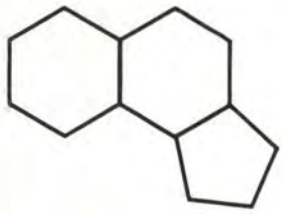
Ex. :



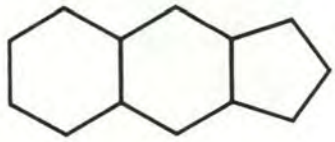
Il suffit de préciser qu'un cycle à 6 atomes est lié à un cycle à cinq atomes et cela par la mise en commun de deux atomes.

1.1.1 Choix d'une représentation par triplets de cycles

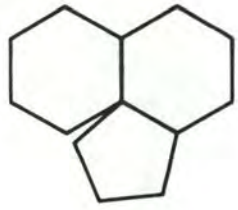
Le problème devient tout de suite plus complexe dans l'exemple suivant où trois cycles sont présents :



On peut toujours dire qu'un cycle à six est lié à un cycle à cinq et qu'il y a en plus un autre ensemble de deux cycles à six atomes liés de la même façon. Il manque cependant quelque chose d'important. C'est la relation entre les deux cycles extrêmes. Il faut en effet pouvoir distinguer cet ensemble de cycles des trois autres cas suivants :



1



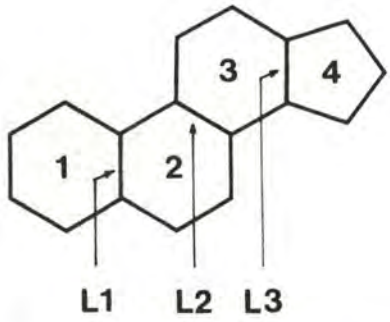
2

3

Donc, si la description des paires de cycles est importante, elle n'est cependant pas suffisante.

On arrive finalement aux problèmes des molécules comportant plus de trois cycles liés les uns aux autres. Nous allons essayer de voir comment on pourrait obtenir une

description complète de la molécule suivante :

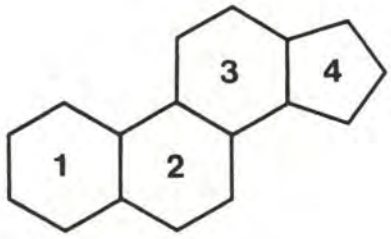


Il faudrait que pour chaque cycle soit défini un nombre suffisant de relations par rapport à d'autres cycles de façon à ce qu'il n'y ait pas d'ambiguïté si l'on désire reconstruire la molécule par la suite. Puisque la description par paires de cycles est, comme nous venons de le signaler, insuffisante, il semble intéressant de se pencher sur une représentation basée sur des ensembles de trois cycles. Dans la molécule tétracyclique ci-dessus, nous pouvons facilement mettre en évidence deux de ces ensembles que nous appellerons triplets :

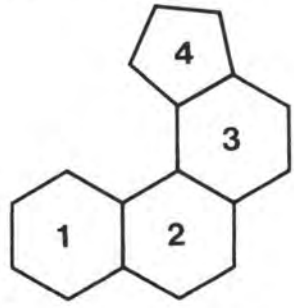
- l'ensemble des cycles 1, 2 et 3
- l'ensemble des cycles 2, 3 et 4.

Considérons tout d'abord le triplet de cycles 1, 2 et 3. Pour le décrire, une idée simple semble être de préciser la position des cycles 1 et 3 sur le cycle 2. Il suffit pour cela de définir les liaisons L1 et L2 qui forment les jonctions entre les cycles du triplet. Si nous définissons une relation entre ces liaisons, cela équivaut à définir une relation entre les cycles et inversément. Nous pouvons alors préciser que ces liaisons sont séparées par une ou trois liaisons appartenant au cycle 2 selon que l'on considère le chemin le plus court ou le plus long entre les liaisons L1 et L2. Il est évident que nous définissons ainsi sans ambiguïté le triplet. Nous pouvons aussi établir le même type de relation pour l'ensemble des cycles 2, 3 et 4. Malheureusement, si la représentation des triplets ne pose

pas de problème, il n'en va plus de même pour les relations entre ces triplets. En effet, avec ces seules descriptions, les deux molécules suivantes sont possibles :



1



2

Et cela est normal puisque nous n'avons en fait, jusqu'à présent, précisé de relations qu'entre les liaisons L1, l2 et L2, L3. Pour lever l'ambiguïté, il suffirait donc de préciser une nouvelle relation entre les liaisons L1 et L3, ce qui établirait ainsi une relation entre les cycles 1 et 4 et, dès lors, une des molécules précédentes ne serait plus possible. Bien sûr, cette relation pourrait de nouveau être précisée par la longueur d'un chemin séparant les liaisons L1 et L3. Mais cela ne risque-t-il pas de nous mener fort loin dans le cas de molécules plus complexes ? Il faudrait donc trouver une possibilité de description des triplets qui permettrait de déterminer en même temps les relations entre ces triplets. Il y a effectivement une façon de procéder de la sorte. Il faudra pour cela donner la description des triplets dans un ordre précis. Nous allons tout d'abord proposer de telles descriptions pour les deux triplets qui nous concernent actuellement. Nous allons ensuite expliquer la façon d'obtenir ces descriptions et voir qu'elles définissent effectivement des relations intéressantes entre les triplets.

cycles 1, 2 et 3

En partant de L2 et en parcourant le cycle 2 dans le sens antihorlogique, on arrive sur L1 après être passé sur une liaison appartenant au cycle 2.

cycles 2, 3, 4

En partant de L2 et en parcourant le cycle 3 dans le sens antihorlogique, on arrive sur L3 après être passé sur une liaison appartenant au cycle 3.

Nous avons donc maintenu une description basée sur le cycle central du triplet, ici les cycles 2 pour le premier et le cycle 3 pour le deuxième. Nous avons ensuite fixé un *sens de parcours* de ces cycles centraux : *antihorlogique*. Ce choix, en lui-même n'est pas important pourvu que le sens de parcours soit toujours le même. Et c'est maintenant qu'intervient une autre convention importante : nous choisissons *le chemin le plus court* entre L1, L2 et L2, L3 pour définir respectivement les premier et deuxième triplets. Nous devons combiner cela avec le sens de parcours antihorlogique du cycle central. C'est ainsi que s'explique ce qui pouvait sembler bizarre dans la description du premier triplet : nous avons pris comme point de départ du parcours du cycle 2 la liaison L2 plutôt que la liaison L1. Cela était nécessaire car en choisissant L1 comme liaison de départ, suite au sens de parcours antihorlogique du cycle 2, nous aurions atteint L2 par le *chemin le plus long*. Pour respecter le sens antihorlogique et le choix du chemin le plus court, il faut donc partir de L2 et aller vers L1.

Il nous faut maintenant trouver une notation pour décrire facilement ce que nous venons de voir. Nous allons tout d'abord choisir de noter un cycle sous la forme

Ct(n)

où C symbolise un cycle
t indique la taille de ce cycle
n est un numéro attribué au cycle (celui du dessin)

Nous allons ensuite représenter une liaison entre deux

cycles par un tiret :

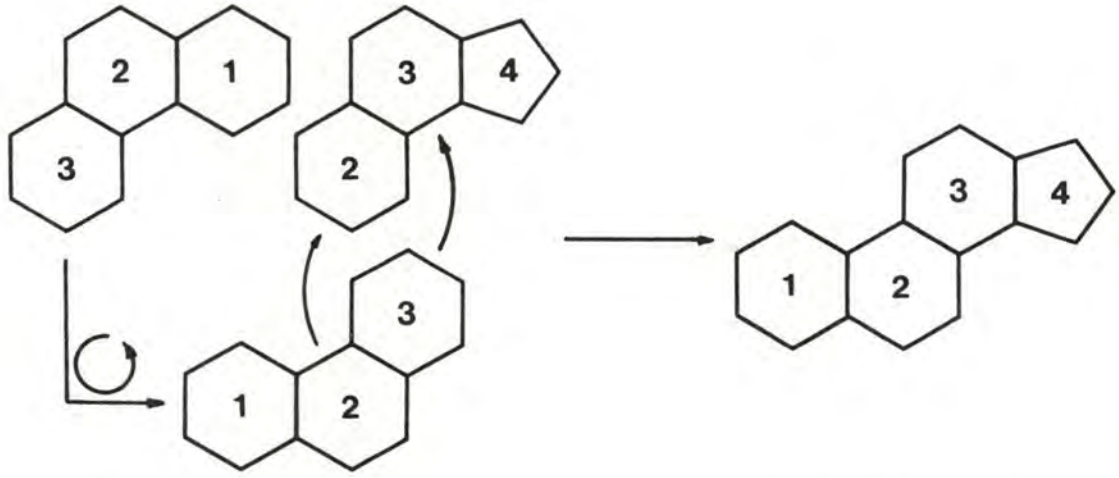
Ct(n) - Ct'(n')

Nous obtenons dès lors la notation suivante pour la molécule dont nous venons de décrire les deux triplets :

C6(3) - C6(2) - C6(1) : 1
C6(2) - C6(3) - C5(4) : 1

où le chiffre de droite indique chaque fois le nombre de liaisons du cycle central séparant celles existant entre les cycles (comme nous venons de le voir).

Il n'y a maintenant plus aucun problème pour reconstituer la molécule en "enchaînant" ces deux lignes de description :

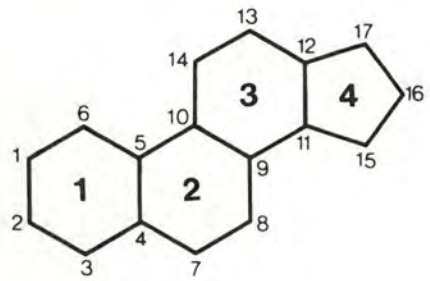


Il est important de remarquer que les triplets ont deux cycles communs entre eux. Ce recouvrement partiel est bien entendu très important car il leur permet de s'orienter l'un par rapport à l'autre. La procédure de reconstruction est dès lors très simple. Il suffit de dessiner les triplets en se basant sur leur description, ce qui ne pose aucun problème. Cela étant fait, il reste alors à les disposer de façon à ce que les cycles identiques (portant le même numéro) puissent se recouvrir.

Nous aborderons plus tard la façon de comparer les représentations de deux molécules. Cela nous permettra de relèver quelques problèmes liés au mode de description que nous venons de décrire. Mais nous allons tout d'abord voir comment obtenir cette description à partir du dessin d'une molécule tel que nous l'avons défini dans le chapitre précédent.

1.1.2 Obtention pratique de la description des cycles

Tout au long de cette partie nous allons nous référer à l'ensemble de cycles suivant, lequel nous a déjà servi pour définir la notion de triplets.



L'obtention des numéros et leur signification effective seront expliqués dans les différentes étapes que nous allons aborder.

1.1.2.1 Recherche des cycles

Rappelons que lors de l'étape de dessin d'une molécule, les cycles sont définis dans une table reprenant leur taille ainsi qu'une séquence de coordonnées de leurs atomes repris selon un ordre de parcours antihorlogique du cycle. Il faut ici remarquer que le choix du parcours antihorlogique pour la description des triplets est simplement lié à ce fait. Mais revenons en à l'exploitation du dessin. Une première

étape consistera à simplifier cette représentation en remplaçant les coordonnées des atomes des différents cycles par un numéro identifiant (deux atomes dans deux cycles différents mais de mêmes coordonnées auront bien sûr le même numéro). De plus, à chaque cycle sera également attribué un numéro identifiant de ce cycle. Nous obtenons ainsi la table suivante :

Description des cycles

C6(1) :	1	2	3	4	5	6
C6(2) :	5	4	7	8	9	10
C6(3) :	10	9	11	12	13	14
C5(4) :	12	11	15	16	17	

Elle reprend tous les numéros des sommets de chaque cycle de notre exemple de référence. C'est à partir de celle-ci que tous les renseignements suivants seront déduits. Cette table contient en effet toute l'information fournie par le dessin sur les cycles et est d'ailleurs tout à fait suffisante pour les reconstruire. Il faut encore faire remarquer que les numéros attribués aux cycles et aux atomes ne doivent pas être considérés de façon "absolue". Ils sont uniquement présents pour différencier les cycles entre eux et les atomes entre eux.

1.1.2.2 Recherche des couples de cycles

La table précédente étant obtenue, il faut maintenant relever tous les couples de cycles. Il nous serons très utiles pour la détermination des triplets. Pour que deux cycles appartiennent au même couple, il suffit qu'ils aient au moins un atome en commun. En fonction des constructions permises durant la phase de dessin de la molécule, le nombre d' atomes communs variera en fait entre un et deux. Il faut également remarquer que dans le cas d'un couple de cycles ayant deux atomes en commun, ceux-ci apparaîtront dans un

ordre inverse dans les deux cycles. On peut facilement s'en rendre compte en observant la table précédente. De plus, ces deux atomes n'apparaîtront pas nécessairement de manière concécutive dans la description si ceux-ci sont les premier et dernier de la liste des atomes du cycle.

La description d'un couple de cycles sera faite de la façon suivante :

Ct(n) - Ct'(n') : S1, S2

où S1 et S2 sont les atomes communs aux deux cycles. Ils sont repris selon l'ordre d'apparition dans le premier cycle de la représentation.

Cela conduit à la table suivante pour notre exemple :

Description des couples de cycles

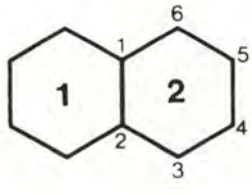
C6(1) - C6(2) :	4, 5
C6(2) - C6(3) :	9, 10
C6(3) - C5(4) :	11, 12

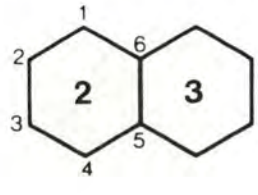
Cette table reprend bien tous les couples de la molécule tels que nous les avons définis.

1.1.2.3 Recherche des triplets de cycles

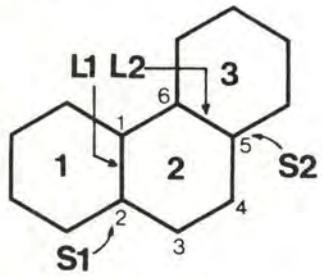
Ces triplets sont simplement composés de deux couples tels qu'un cycle d'un de ces couples soit également présent dans l'autre. Ils décriront toutes les relations possibles entre tous les couples.

Considérons les deux couples de cycles suivants :





Les atomes du cycle 2 sont bien entendu numérotés de la même façon dans les deux couples. Il est donc aisé de construire le triplet correspondant.



Si nous prenons le triplet dans cet ordre, le comptage des liaisons sur le cycle central (2) séparant les cycles extrêmes (1 et 3) commencera par la liaison L1 des cycles 1 et 2. Rappelons que le cycle central doit être parcouru dans le sens antihorlogique. La première liaison à considérer sur le chemin L1 - L2 sera donc celle commençant par l'atome S1 et la dernière, celle se terminant par S2. Il est alors simple de voir si ce chemin est le plus court. Si c'est le cas, on doit avoir la relation :

Long *inférieur à* (t2 - NL) / 2

où Long est la longueur du chemin trouvé
t2 est la taille du cycle central
NL est le nombre de liaisons du cycle central partagées avec les cycles extrêmes.

En cas d'égalité, on pourrait éventuellement ordonner le triplet en fonction de la taille des cycles et de la nature des liaisons. Il s'agit là justement d'un des problèmes que nous aborderons plus tard lorsque nous verrons comment comparer deux descriptions de molécule. Pour en revenir à notre exemple, la longueur du chemin est supérieure à (t2 - NL) / 2. Il faudra donc inverser les cycles 1 et 3 dans la description du triplet et la longueur du chemin devient (t2 - NL - Long). Nous respectons bien ainsi les conventions que nous avons définies précédemment.

L'ensemble des triplets est facilement obtenu en prenant chaque couple de cycles et en parcourant la liste des autres couples qui le suivent. Un de ces derniers est retenu s'il présente un cycle identique à l'un de ceux du premier couple.

Nous présentons ci-dessous la table de triplets obtenue pour notre molécule de référence.

Description des triplets de cycles

C6(3) - C6(2) - C6(1) distance : 1 liaisons : 2, 2
C6(2) - C6(3) - C5(4) distance : 1 liaisons : 2, 2

Dans cette table, le nombre qui suit "distance" représente la longueur du chemin le plus court comme nous l'avons défini. Les deux nombres de la partie "liaisons" reprennent simplement le type de liaison respectivement entre les premier et deuxième cycles et deuxième et troisième cycles du triplet : chaque liaison entre cycles fait bien intervenir deux atomes. Comme nous pouvons le constater, les numéros des sommets ont finalement disparus dans cette représentation. Quant aux numéros de cycles, rappelons qu'ils n'ont aucune signification propre autre que celle de permettre d'établir les relations entre les triplets d'une même molécule.

1.1.2.4 Recherche des groupes de triplets

Dans une molécule, tous les cycles ne sont pas nécessairement reliés entre eux. Les triplets peuvent donc être scindés en plusieurs groupes tels qu'aucun des cycles d'un groupe ne se retrouve dans un autre groupe. Il serait donc intéressant que les descriptions de ces groupes ne se retrouvent pas imbriquées les unes dans les autres. Une façon simple de remédier à ce problème est de déjà trier la

liste de couples en fonction de ce même critère. Ce tri est nécessaire car le dessin des cycles n'a pas nécessairement été réalisé groupe après groupe. Ce tri permettra d'autre part de détecter les couples de cycles isolés (n'intervenant dans aucun triplet) et réduira aussi la liste des paires de cycles à parcourir dans la recherche des triplets. En effet, pour déterminer un triplet, il suffira de parcourir uniquement les couples de cycles du groupe auquel il appartient.

Remarque : Il est également possible de détecter les cycles isolés lors de la recherche des couples de cycles. Il suffit simplement de "marquer" un cycle chaque fois qu'il est repris dans un couple. Les cycles isolés seront simplement ceux qui ne sont pas marqués en fin de recherche.

1.1.3 Comparaison de deux descriptions

Ces comparaisons seront bien sûr inévitables pour rechercher si une molécule est présente dans la base de données. Nous allons aborder les comparaisons de manière à mettre en évidence deux problèmes essentiels liés à la description des cycles sous forme de triplets. Il s'agit de

- l'influence d'une rotation du dessin de la molécule sur la description
- la description des triplets "symétriques" pour lesquels il n'y a pas de "chemin le plus court".

1.1.3.1 Influence de l'orientation du dessin

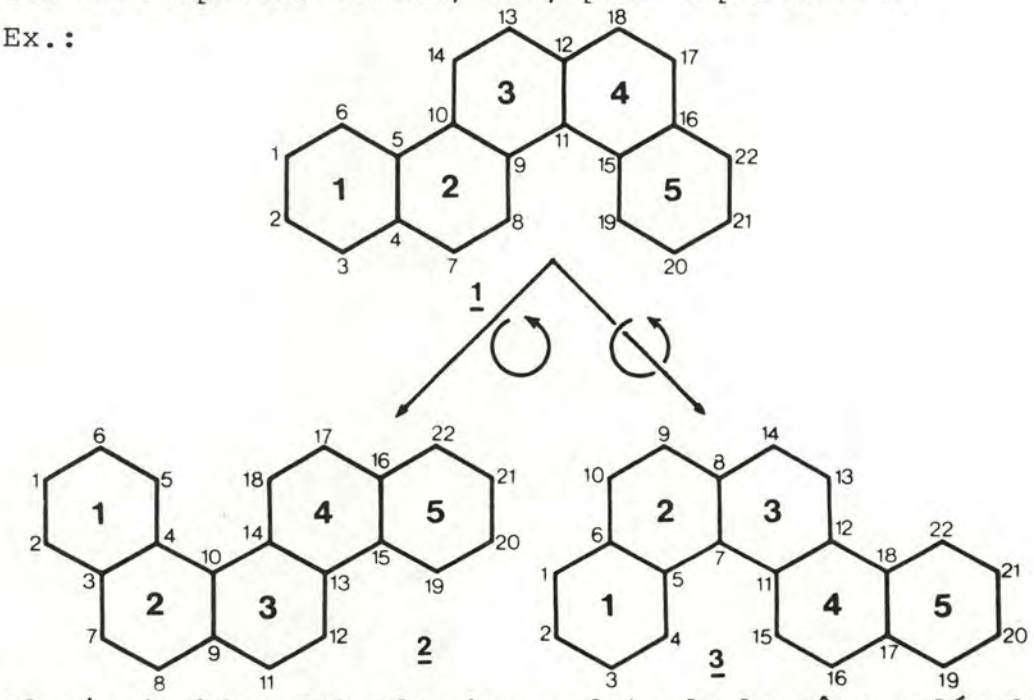
Il semble difficile d'imposer à l'utilisateur d'employer une orientation "standard" pour dessiner les molécules. D'ailleurs, comment pourrait-on définir une telle

orientation ? Il faudra donc essayer de rendre la description d'une molécule "insensible" à une rotation du dessin. On se rend vite compte que deux types de rotations sont à considérer :

celles dans le plan du dessin qui ne devraient normalement influencer que l'ordre de la description

celles en dehors du plan qui donnent deux dessins images l'un de l'autre dans un miroir. Les influences sur les descriptions seront, ici, plus importantes.

Ex.:



Il s'agit bien entendu chaque fois de la même molécule. Nous allons maintenant observer les descriptions obtenues pour chacun de ces dessins. Celles-ci sont décrites à la page suivante. Seule, ici, la description des triplets de cycles va réellement nous intéresser.

De toute évidence, la différence entre les deux premiers dessins résulte d'une simple rotation *dans le plan*. A priori, les descriptions de ces deux dessins semblent pourtant différentes, que ce soit au niveau des numéros de

DESSIN 1

Description des cycles

```
C6( 1) :    1    2    3    4    5    6
C6( 2) :    5    4    7    8    9   10
C6( 3) :   10    9   11   12   13   14
C6( 4) :   12   11   15   16   17   18
C6( 5) :   16   15   19   20   21   22
```

Description des couples de cycles

```
C6( 1) - C6( 2) :    4,   5
C6( 2) - C6( 3) :    9,  10
C6( 3) - C6( 4) :   11,  12
C6( 4) - C6( 5) :   15,  16
```

Description des triplets de cycles

```
C6( 3) - C6( 2) - C6( 1)   distance : 1   liaisons : 2, 2
C6( 2) - C6( 3) - C6( 4)   distance : 1   liaisons : 2, 2
C6( 3) - C6( 4) - C6( 5)   distance : 1   liaisons : 2, 2
```

DESSIN 2

Description des cycles

```
C6( 1) :    1    2    3    4    5    6
C6( 2) :    4    3    7    8    9   10
C6( 3) :   10    9   11   12   13   14
C6( 4) :   14   13   15   16   17   18
C6( 5) :   16   15   19   20   21   22
```

Description des couples de cycles

```
C6( 1) - C6( 2) :    3,   4
C6( 2) - C6( 3) :    9,  10
C6( 3) - C6( 4) :   13,  14
C6( 4) - C6( 5) :   15,  16
```

Description des trplets de cycles

```
C6( 3) - C6( 2) - C6( 1)   distance : 1   liaisons : 2, 2
C6( 4) - C6( 3) - C6( 2)   distance : 1   liaisons : 2, 2
C6( 3) - C6( 4) - C6( 5)   distance : 1   liaisons : 2, 2
```

DESSIN 3

Description des cycles

```
C6( 1) :    1    2    3    4    5    6
C6( 2) :    6    5    7    8    9   10
C6( 3) :    8    7   11   12   13   14
C6( 4) :   12   11   15   16   17   18
C6( 5) :   18   17   19   20   21   22
```

Description des couples de cycles

```
C6( 1) - C6( 2) :    5,   6
C6( 2) - C6( 3) :    7,   8
C6( 3) - C6( 4) :   11,  12
C6( 4) - C6( 5) :   17,  18
```

Description des triplets de cycles

```
C6( 1) - C6( 2) - C6( 3)   distance : 1   liaisons : 2, 2
C6( 2) - C6( 3) - C6( 4)   distance : 1   liaisons : 2, 2
C6( 5) - C6( 4) - C6( 3)   distance : 1   liaisons : 2, 2
```

certains atomes ou de l'ordre d'apparition des numéros de cycles dans les triplets. Nous allons voir qu'il n'en est rien. Tout d'abord, les numéros des atomes ont disparu dans les triplets. Quant aux numéros de cycles, nous avons déjà dit qu'ils n'avaient d'intérêt qu'*à l'intérieur d'une même molécule*. Deux molécules tout à fait identiques peuvent très bien avoir des numéros de cycles différents. Nous allons donc voir comment la comparaison peut-être faite.

Considérons le premier triplet de la description de la molécule 1. Il devrait normalement avoir son équivalent dans la description de la deuxième molécule puisqu'elles sont théoriquement identiques. Ne nous occupons tout d'abord pas des numéros des cycles. Il suffit d'avoir trois cycles de six atomes et une distance de une liaison. Nous devons de même trouver dans la partie "liaisons" les nombres 2,2. Nous avons le choix entre les trois triplets qui répondent tous à ces conditions. Prenons le premier. Regardons ensuite le deuxième triplet de la première description. Nous devons maintenant prendre les numéros de cycles en compte. En effet, ceux-ci nous renseignent sur les relations entre les triplets d'une même molécule. Nous voyons donc que ce deuxième triplet présente la relation suivante avec le premier : les cycles en première et deuxième position du premier triplet se retrouvent respectivement en deuxième et première position dans le deuxième triplet. Cette relation doit donc exister aussi pour la deuxième description. Etant donné le choix du premier triplet que nous avons fait, nous devrions normalement découvrir un triplet dont les deux premiers cycles portent les numéros 2 et 3. Un tel triplet n'existe pas. C'est que notre premier choix était faux. Nous devons donc le revoir en prenant le deuxième triplet de la deuxième description comme équivalent au triplet un de la première. Après un nouvel échec, nous obtiendrons finalement, la solution représentée par la série d'équivalences suivantes :

triplets description 1	triplets description 2
1	3
2	2
3	1

où les relations entre les triplets 1 et 2 et les triplets 2 et 3 de la première description se retrouvent respectivement entre les triplets 3, 2 et 2, 1 de la deuxième.

A partir de là, on peut éventuellement faire apparaître les correspondances entre les numéros cycles des deux premiers dessins :

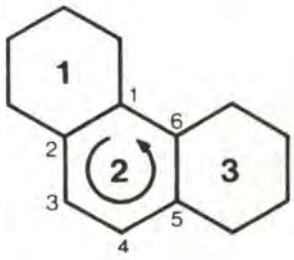
cycles dessins 1	cycles dessins 2
1	5
2	4
3	3
4	2
5	1

ce que l'on vérifie effectivement, et heureusement, si l'on superpose les deux premiers dessins.

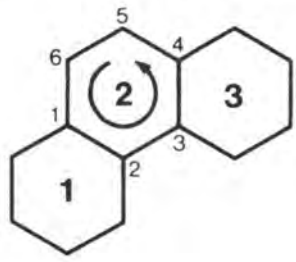
Comme nous venons de le voir, une rotation dans le plan n'est absolument pas génante pour la comparaison. Et c'est tout à fait normal puisque le parcours dans le sens antihorlogique du cycle central du triplet est indépendant d'une rotation dans le plan de ce triplet.

Il en va cependant autrement dans le cas d'une rotation en dehors du plan du dessin. Observons les dessins 2 et 3 de la molécule, et plus particulièrement, les cycles 1, 2 et 3 qui sont, de toute évidence, les mêmes dans la molécule. La conséquence de la rotation sur ce triplet de cycles va être une permutation des cycles extrêmes dans l'ordre de la description. Cela est logique puisque la rotation a pour

effet d'inverser l'ordre de parcours des atomes du cycle 2 pour maintenir le sens antihorlogique de ce parcours.



C6(3) - C6(2) - C6(1) : 1



C6(1) - C6(2) - C6(3) : 1

Cela n'aurait pas d'influence dans la description d'un triplet isolé, mais ce n'est pas le cas ici. Observons donc que deux cycles qui jouent le même rôle dans les deux dessins ont été inversés dans la description suite à la rotation hors du plan. Ce problème ne peut bien sûr être éliminé au moment de la création de la description puisqu'il est impossible de savoir si la molécule a subi une rotation. Sur quelle base pourrait-on en effet se reposer pour le déterminer. Ce ne sera donc qu'au moment de la comparaison que l'on pourra déterminer si deux molécules sont séparées l'une de l'autre par une rotation hors du plan. En effet, dans ce cas, la comparaison précédemment décrite échouera. Si c'est le cas, il suffit simplement d'inverser les triplets de cycles d'une des deux molécules (permuter les cycles extrêmes) et de recommencer les tests. S'il y a de nouveau échec, c'est que les deux molécules sont vraiment différentes. Prenons donc comme exemple les dessins 2 et 3 de la molécule précédente de cinq cycles. On s'aperçoit tout de suite que la comparaison échoue. On trouve cependant une relation identique entre deux triplets. Les cycles 1, 2, 3 et 4 des deux dessins sont en cause. Cela est simplement dû au fait que ces cycles forment un motif symétrique et sont donc insensibles à une rotation hors du plan. Mais revenons à notre problème. Si on inverse

les triplets de cycles de la troisième description, par exemple, il n'y aura plus de problème. En effet, cette dernière devient alors :

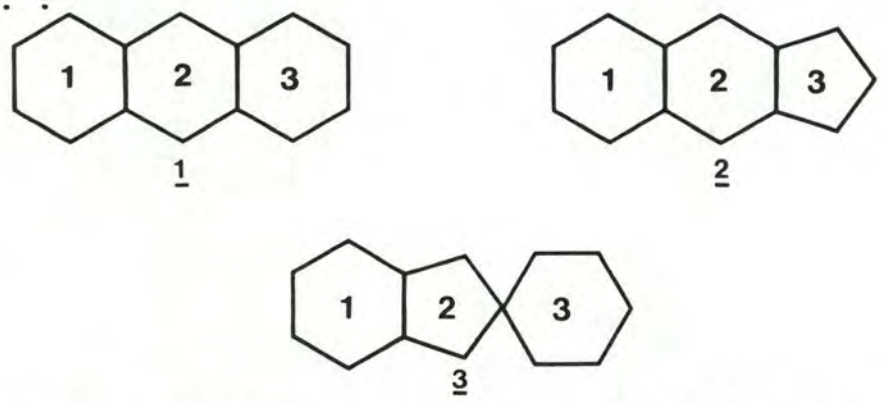
1. C6(3) - C6(2) - C6(1) : 1
2. C6(4) - C6(3) - C6(2) : 1
3. C6(3) - C6(4) - C6(5) : 1

et les lignes 1, 2 et 3 de la deuxième description s'identifient respectivement aux lignes 1, 2 et 3 ci-dessus.

1.1.3.2 Problème des triplets symétriques

Il reste encore un problème à soulever concernant cette représentation sous forme de triplets. Il existe des agencements de cycles tels qu'il ne sera pas possible de définir un ordre précis pour ces cycles.

Ex. :



Il n'y a en effet pas plus de raisons de définir le premier triplet de la molécule 1 comme

C6(1) - C6(2) - C6(3) : 2

plutôt que

C6(3) - C6(2) - C6(1) : 2

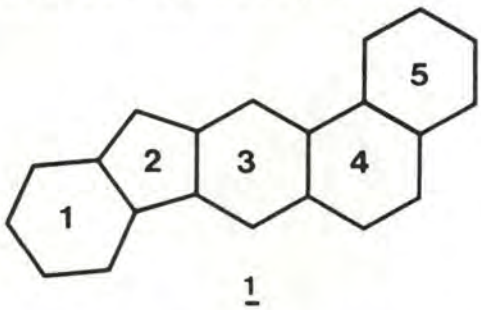
puisque la distance entre les deux cycles extrêmes par rapport au cycle central est la même quel que soit le cycle origine.

Ce sera de nouveau au moment des comparaisons qu'une

telle ambiguïté devra être levée. Lorsqu'un tel triplet symétrique sera reconnu dans une des molécules au moment d'une comparaison, il suffira simplement de rechercher la présence d'une des deux formes de ce triplet dans l'autre molécule.

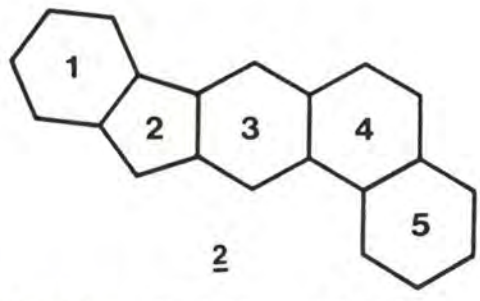
Remarque : Dans le cas des dessins 2 et 3, on ne peut pas à proprement parler de triplets symétriques puisque soit les cycles extrêmes ne sont pas de même taille soit les liaisons de ces cycles avec le cycle central sont de nature différente. Pour ceux-ci, il y aurait donc une possibilité de les décrire dans un certain ordre mais indépendant de l'orientation du triplet dans la molécule. De tels triplets devront donc toujours être traités comme des cas particuliers s'il faut comparer deux dessins d'une même molécule ayant subit une rotation en dehors du plan l'un par rapport à l'autre. En effet,dans ce cas, il faudra inverser les triplets de cycles d'une des représentations sauf ceux représentant les groupes de cycles que nous venons d'évoquer :

Dans l'exemple suivant, nous avons choisi de placer le cycle de plus petite taille à gauche de la description des triplets "symétriques".



1

C6(1) - C5(2) - C6(3) : 1
C5(2) - C6(3) - C6(4) : 2
C6(5) - C6(4) - C6(3) : 1



2

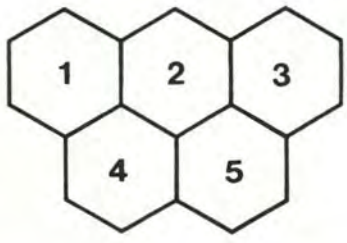
C6(3) - C5(2) - C6(1) : 1
C5(2) - C6(3) - C6(4) : 2
C6(3) - C6(4) - C6(5) : 1

De toute évidence, les comparaisons devront nécessiter la permutation d'un des groupes de triplets pour montrer l'identité des deux dessins. Il ne faudra cependant pas inverser la deuxième ligne puisque celle-ci est ordonnée indépendemment d'une rotation hors plan de la molécule.

Il peut cependant sembler plus général de ne pas introduire de tel cas particulier et de traiter tous les triplets symétriques de la même manière quelle que soit la taille des cycles et la nature de leurs liaisons. Nous renforcerons encore plus loin cet état de chose.

1.1.3.3 Simplification de la description

Dans certains cas d'agencement de cycles, la liste des triplets peut devenir longue. Considérons en effet la molécule suivante dont la description des triplets suit.




```
C6( 4) - C6( 1) - C6( 2)   : 0
C6( 1) - C6( 2) - C6( 3)   : 2
C6( 1) - C6( 2) - C6( 4)   : 0
C6( 1) - C6( 2) - C6( 5)   : 1
C6( 2) - C6( 4) - C6( 1)   : 0
C6( 5) - C6( 4) - C6( 1)   : 1
C6( 4) - C6( 2) - C6( 3)   : 1
C6( 5) - C6( 2) - C6( 3)   : 0
C6( 2) - C6( 3) - C6( 5)   : 0
C6( 4) - C6( 2) - C6( 5)   : 0
C6( 5) - C6( 4) - C6( 2)   : 0
C6( 3) - C6( 5) - C6( 2)   : 0
C6( 2) - C6( 5) - C6( 4)   : 0
C6( 3) - C6( 5) - C6( 4)   : 1
```

Si beaucoup de ces lignes sont redondantes pour décrire de façon non ambiguë la molécule, elles sont cependant nécessaires si l'on désire rechercher dans la molécule tel ou tel ensemble de cycles. Mais il en va tout autrement pour rechercher la présence de cette structure de cycles dans une autre description. Il serait dès lors intéressant de ne retenir qu'un ensemble minimum de ces triplets, cet ensemble devant caractériser entièrement la structure de cycles. Pour cela, la liste de triplets retenue doit simplement répondre aux conditions suivantes :

1. chaque cycle doit être présent au moins une fois dans l'ensemble choisi,

2. pour chaque triplet de la liste, deux de ses cycles successifs doivent se retrouver, également l'un à la suite de l'autre, dans un autre triplet de la liste. Le troisième cycle ne peut déjà être présent dans la liste.

Si nous modifions la deuxième condition de la façon suivante : "pour chaque triplet de la liste sauf le premier, ... dans un autre triplet *le précédent* dans la liste ...", nous obtenons une autre propriété très intéressante. En effet, jusque maintenant, quand nous avons développé les comparaisons, nous avons toujours supposé qu'il existait une relation entre deux triplets successifs. Il est évident que ce ne sera pas toujours le cas, les triplets n'étant pas définis dans un ordre précis. Dans la liste que nous allons obtenir en appliquant les deux conditions, nous aurons toujours à rechercher un triplet dont on connaît déjà deux cycles.

Plusieurs listes différentes peuvent bien entendu être ainsi définies. Le choix entre celles-ci n'a pas d'importance pour la réussite de la comparaison avec une autre molécule puisque, si cette dernière reprend bien la

même structure de cycles, sa description contient obligatoirement tous les triplets définissant cette structure. Il faut également remarquer que le nombre de triplets dans chacune des listes sera toujours le même puisque tous les triplets ajoutés au premier de la liste n'introduisent qu'un cycle supplémentaire. Le nombre de triplets sera donc égal à la somme des cycles moins deux. Voici une des listes possibles :

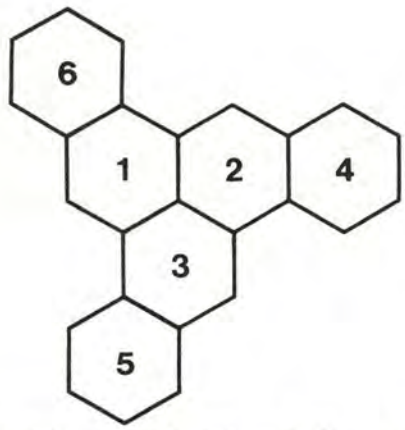
```
1.  C6( 4) - C6( 1) - C6( 2)  distance: 0  liaisons: 2,2
                              réf.:     pos.:
2.  C6( 1) - C6( 2) - C6( 3)  distance: 2  liaisons: 2,2
                              réf.: 1   pos.: 1,2 2,3
3.  C6( 1) - C6( 2) - C6( 5)  distance: 1  liaisons: 2,2
                              réf.: 1   pos.: 1,2 2,3
```

Voyons un peu ce que représentent ces lignes. La partie "réf." (référence) de chaque triplet indique à quel autre triplet de la liste il est relié. Il est normal que la ligne numéro 1 ne se réfère à aucune autre puisqu'elle a été la première choisie. La partie "pos." nous renseigne sur la position des cycles communs de la ligne avec la ligne référencée. Par exemple, les cycles aux positions 1 et 2 du deuxième triplet se retrouvent respectivement aux positions 2 et 3 du premier triplet. Simplement, la recherche des triplets ne se fera plus maintenant par l'intermédiaire de la relation entre deux triplets successifs mais bien grâce à une relation entre deux triplets quelconques (mais connus).

Nous avons précédemment évoqué le problème lié aux triplets "symétriques". Pour certaines structures de cycles, telle celle que nous venons de voir, il aurait été possible d'éviter la présence de triplets de ce type dans la liste minimale de triplets décrivant les cycles. Cela risque

cependant d'être lourd comme recherche. En effet, un triplet n'est ajouté à la liste que s'il répond à la condition 2 décrite plus haut. Or, il pourrait arriver qu'il ne soit plus possible d'ajouter, à un moment donné, d'autres triplets qu'un qui soit symétrique. Il faudra donc revoir les choix précédents jusqu'à trouver une solution sans triplet symétrique si c'est possible, ce qui est difficile à détecter à priori.

Ex. :




```
C6( 3) - C6( 1) - C6( 2)  : 0  liaisons : 2, 2
C6( 2) - C6( 1) - C6( 6)  : 1  liaisons : 2, 2
C6( 1) - C6( 2) - C6( 3)  : 0  liaisons : 2, 2
C6( 1) - C6( 2) - C6( 4)  : 2  liaisons : 2, 2
C6( 3) - C6( 1) - C6( 6)  : 2  liaisons : 2, 2
C6( 2) - C6( 3) - C6( 1)  : 0  liaisons : 2, 2
C6( 1) - C6( 3) - C6( 5)  : 1  liaisons : 2, 2
C6( 3) - C6( 2) - C6( 4)  : 1  liaisons : 2, 2
C6( 2) - C6( 3) - C6( 5)  : 2  liaisons : 2, 2
```

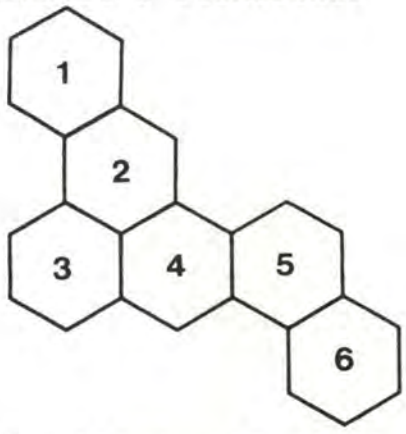
Ce n'est qu'après avoir essayé toutes les possibilités de description des cycles 1, 2 et 3 de cette molécule que l'on s'apercevra qu'il n'est pas possible d'établir une liste minimale ne comportant que des triplets non symétriques.

Nous devons encore parler d'un problème lié aux triplets symétriques, qu'ils soient composés de cycles de même taille ou non. Ce problème va une nouvelle fois influencer l'exécution des comparaisons de structures de cycles. En effet, contrairement aux triplets non symétriques, il peut arriver, lors d'une comparaison, d'avoir le choix entre plusieurs triplets symétriques. Il faudra donc d'abord revenir à ces choix si une comparaison

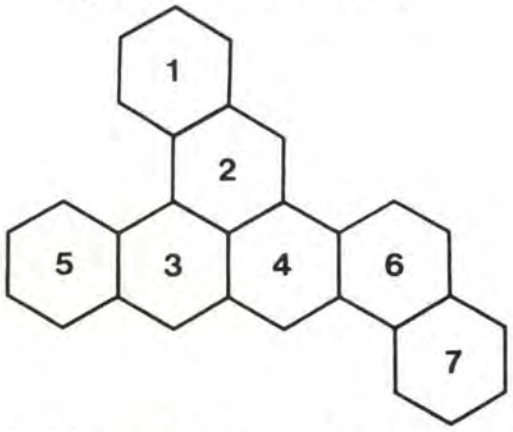
échoue, plutôt que de la recommencer en prenant simplement un nouveau triplet initial comme nous l'avons fait précédemment.

Ex. :

molécule à rechercher



molécule candidate



C6(1) - C6(2) - C6(3) : 1
C6(4) - C6(3) - C6(2) : 0
C6(3) - C6(4) - C6(5) : 2
C6(4) - C6(5) - C6(6) : 1

.1. C6(1) - C6(2) - C6(3) : 1
.2. C6(1) - C6(2) - C6(4) : 2
.3. C6(3) - C6(2) - C6(4) : 0
.4. C6(4) - C6(3) - C6(2) : 0
5. C6(2) - C6(3) - C6(5) : 1
6. C6(2) - C6(4) - C6(3) : 0
7. C6(6) - C6(4) - C6(2) : 1
8. C6(4) - C6(3) - C6(5) : 2
9. C6(3) - C6(4) - C6(6) : 2
10. C6(4) - C6(6) - C6(7) : 1

Aux deux premières lignes de la molécule à rechercher, nous pouvons faire correspondre les premier et quatrième triplets de la description de la molécule candidate. Il y a maintenant deux choix possibles pour la troisième ligne. Si nous prenons le huitième triplet de la molécule candidate, nous aboutirons à un échec. Il faut donc d'abord analyser l'autre possibilité avant de tenter un autre choix pour la première ligne de la molécule à rechercher (comme on l'aurait directement fait dans le cas où aucun de ses triplets n'aurait été symétrique). Nous essayons donc le neuvième triplet de la molécule candidate. Ce choix se révèlera fructueux puisqu'il conduit à la réussite de la comparaison. Il est donc très important de mémoriser chaque endroit où un triplet symétrique a été choisi pour pouvoir y revenir en cas d'échec.

Remarque : Il est aisé de se rendre compte que dans le cas de triplets non symétriques, on ne peut avoir, à chaque instant, au plus qu'un seul de ces triplets candidat. De fait, lors du choix d'un triplet, deux cycles et leurs positions sont fixés. Par ailleurs, la distance entre les deux cycles extrêmes du triplet ainsi que la taille du troisième cycle sont également connus. Il ne peut de la sorte être défini qu'un seul triplet puisque, par construction, on ne peut avoir deux cycles de même taille occupant la même position sur le dessin.

1.2 Description des chaînes

Cette partie de la description a pour but de mettre en évidence les différentes chaînes contenues dans une molécule en considérant, comme pour la description des cycles, que tous les atomes sont identiques.

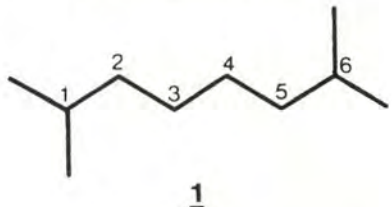
Rappelons que toutes les chaînes doivent avoir au moins un atome commun avec un cycle, que tous les groupes de cycles doivent normalement être reliés entre eux par une chaîne (si deux groupes de cycles étaient reliés par plus d'une chaîne, cet ensemble formerait un cycle non détecté, ce qui est interdit par construction) et enfin, que si une molécule ne contient pas de cycle, elle ne peut normalement être composée que d'une seule chaîne. Nous supposons donc que la molécule ne forme qu'un seul bloc.

1.2.1 Méthode de description

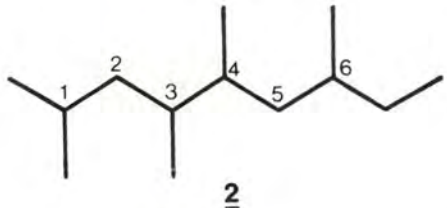
En chimie, une chaîne quelconque est définie en précisant d'abord la chaîne la plus longue. Les autres fragments sont ensuite décrits ainsi que les atomes de la

chaîne précédemment définie sur lesquels ils sont liés.

Nous avons préféré choisir une méthode plus simple, évitant d'avoir à rechercher la chaîne la plus longue. Elle consiste à énumérer tous les fragments de chaîne compris entre les atomes au moins trisubstitués ou terminaux. Il faut cependant bien se rendre compte que, quelle que soit la méthode de codage choisie, des comparaisons "complexes" entre chaînes ne pourront être réalisées que si ces dernières sont considérées comme des arbres. Il n'est en effet pas simple de calculer si la chaîne 1 est présente dans la chaîne 2 :



1

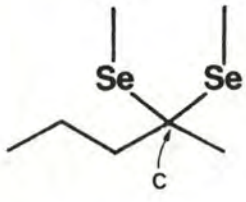


2

Il semble donc que l'une ou l'autre méthode ne soient intéressantes que pour montrer si deux chaînes sont identiques. Les autres types de comparaisons se montreront vite très lourds.

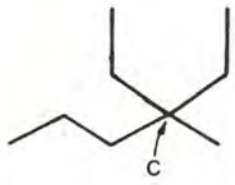
1.2.2 Intérêt de la description des chaînes

La description des chaînes peut quand-même être intéressante pour confirmer le choix d'une molécule. Supposons que nous recherchions la molécule suivante :



Un premier choix se portera bien entendu sur les molécules

contenant une fonction "acétal sélénié" (entourée sur le dessin). La sélection parmi les molécules candidates pourra alors être réalisée par la recherche de la chaîne suivante dans la description des mêmes molécules :



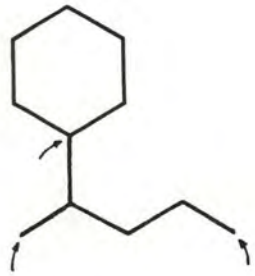
Des tests supplémentaires peuvent encore être faits pour vérifier que l'atome C de l'acétal correspond bien à l'atome C de la chaîne précédente. Il pourrait éventuellement en être fait de même avec les atomes de sélénium. Cependant, ce sommet étant disubstitué, il n'apparaît pas dans la description des chaînes. Un tel test semble donc beaucoup plus délicat.

1.2.3 Obtention pratique de la description des chaînes

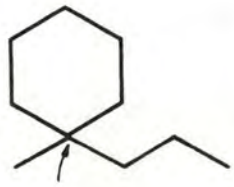
Il faudra essentiellement identifier les fragments de chaînes séparés par les atomes au moins trisubstitués et isoler les descriptions de ces mêmes chaînes.

Comme pour les cycles, une première étape consistera à remplacer les coordonnées des atomes par un numéro identifiant en tenant évidemment compte de ceux déjà attribués aux atomes des cycles.

La description est alors obtenue en considérant cet ensemble d'atomes liés comme un arbre. Le premier atome sera choisi parmi ceux possédant un degré de substitution 1 dans la chaîne. Dans l'exemple suivant, ils sont indiqués par une flèche :

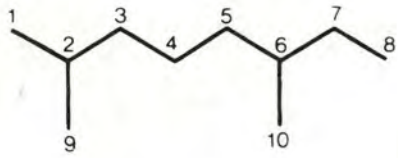


Du fait de la description séparée des chaînes et des cycles, il faudra tenir compte du cas particulier visible dans l'exemple suivant :



Dans cet exemple, on a en fait deux chaînes branchées sur le même atome du cycle. Or, pratiquement, elles seront vues comme une seule chaîne composée uniquement d'atomes mono et disubstitués. Avant d'identifier les chaînes, il sera donc important d'isoler tous les atomes disubstitutés, de vérifier s'ils appartiennent aussi à un cycle et, si c'est le cas, d'en augmenter le degré de substitution.

Voici maintenant un exemple de description d'une chaîne :



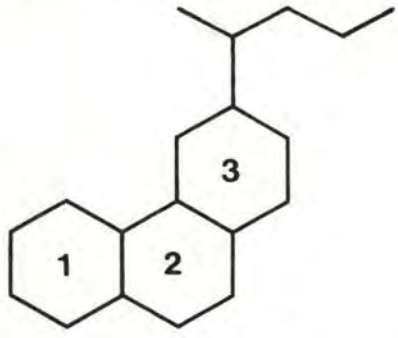
premier atome : 1 - longueur : 2 - dernier atome : 2
premier atome : 2 - longueur : 5 - dernier atome : 6
premier atome : 6 - longueur : 3 - dernier atome : 8
premier atome : 6 - longueur : 2 - dernier atome : 10
premier atome : 2 - longueur : 2 - dernier atome : 9

Les termes "premier atome" et "dernier atome" désignent les

atomes extrémités des chaînes. La longueur indique le nombre d'atomes (y compris les atomes délimitant la chaîne) dans la chaîne désignée.

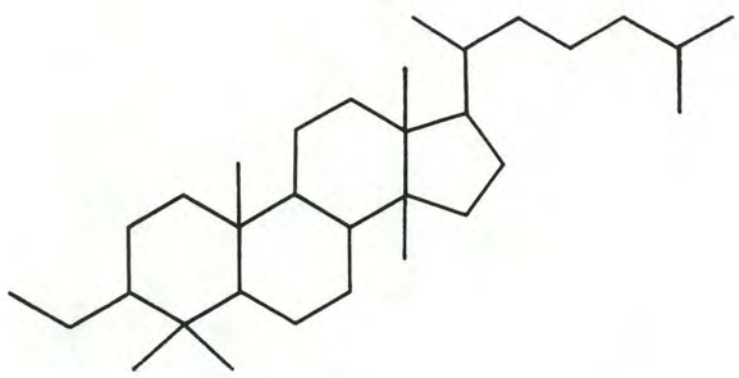
1.3 Relation entre les descriptions du squelette

Nous allons tout d'abord présenter la description complète d'une molécule telle que nous l'avons vue jusqu'ici. Elle est présentée à la page suivante. Cette description permet de reconstruire sans problème le dessin : les groupes de cycles sont reconstruits à l'aide de la représentation sous forme de triplets, les différentes chaînes pouvant elles aussi être aisément reformées. Ce qui pose plus de problèmes, ce sont les relations existant entre les chaînes et les cycles. A ce stade, il faut en effet retourner à la description de chaque cycle pour connaître la façon dont les chaînes y sont liées. D'où une idée serait d'ajouter un nouveau type de triplets particuliers qui couvrirait les relations entre les cycles et les chaînes. Voyons cela dans l'exemple suivant :



soit le triplet C6(3) - C6(2) - C6(1) : 1 décrivant l'ensemble des cycles et une description "DC1" représentant la chaîne branchée sur le cycle 3. La relation entre cycle et chaîne serait alors traduite par le triplet suivant :

DC1 - C6(3) - C6(2) : 2



Description des cycles

```
C6( 1) :    1    2    3    4    5    6
C6( 2) :    5    4    7    8    9   10
C6( 3) :   10    9   11   12   13   14
C5( 4) :   12   11   15   16   17
```

Description des couples de cycles

```
C6( 1) - C6( 2) :    4,   5
C6( 2) - C6( 3) :    9,  10
C6( 3) - C5( 4) :   11,  12
```

Description des triplets de cycles

```
C6( 3) - C6( 2) - C6( 1)   distance : 1   liaisons : 2, 2
C6( 2) - C6( 3) - C5( 4)   distance : 1   liaisons : 2, 2
```

Description des chaines

```
premier atome :  2  -  longueur : 3  -  dernier atome :  32
                          ******
premier atome :  5  -  longueur : 2  -  dernier atome :  28
                          ******
premier atome : 11  -  longueur : 2  -  dernier atome :  27
                          ******
premier atome : 12  -  longueur : 2  -  dernier atome :  26
                          ******
premier atome : 17  -  longueur : 2  -  dernier atome :  18
premier atome : 18  -  longueur : 2  -  dernier atome :  19
premier atome : 18  -  longueur : 5  -  dernier atome :  23
premier atome : 23  -  longueur : 2  -  dernier atome :  24
premier atome : 23  -  longueur : 2  -  dernier atome :  25
                          ******
premier atome : 29  -  longueur : 2  -  dernier atome :   3
premier atome :  3  -  longueur : 2  -  dernier atome :  30
```

Ce genre de relation peut également être utilisée pour définir les positions de plusieurs chaînes sur un même cycle.

Ces relations sont obtenues selon une procédure tout à fait semblable à celle utilisée pour les triplets de cycles. Une première étape consistera à relever tous les couples *cycle - chaîne*. Il faut ensuite passer en revue tous les couples et ne retenir que les triplets dont l'élément central est un cycle et dont, bien entendu, au moins un des éléments extrêmes est une chaîne.

2. DESCRIPTION DES FONCTIONS CHIMIQUES

Les fonctions ont été introduites lors de l'étape de dessin par modification du squelette en y identifiant certains atomes par un symbole chimique ou en y ajoutant des liaisons multiples entre atomes. Il ne sera donc pas possible, comme pour les cycles, d'identifier et isoler simplement les fonctions chimiques.

2.1 Reconnaissance des fonctions

2.1.1 Isolement des fonctions

La première chose à faire sera donc de mettre en évidence les atomes composant une fonction. Nous allons pour cela considérer deux types de fonctions : celles faisant intervenir au moins un hétéroatome et celles constituées uniquement par des liaisons multiples entre carbones non reprises dans les premières fonctions. Il va donc falloir définir un ensemble de règles pour déterminer quelle

fraction de l'entourage d'un hétéroatome appartient à la fonction à laquelle il participe.

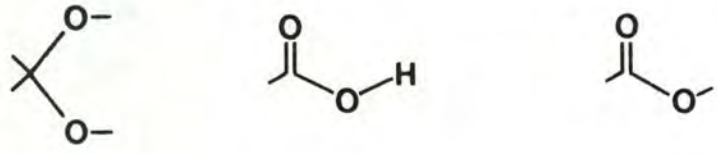
2.1.1.1 Fonctions possédant un hétéroatome

Dans cette étape, la recherche se fera donc à partir des hétéroatomes introduits dans la molécule. Cela permettra de ne localiser la recherche que sur certains points de cette molécule.

Toute fonction sera donc initialement composée d'un hétéroatome. Nous allons tenter de proposer quelques règles pour "développer" cette fonction. Elles définiront les conditions à remplir par un atome pour qu'il soit ajouté à la fonction actuellement développée.

(1) Tout hétéroatome séparé par au plus un atome de carbone, appartenant ou non à la fonction, d'un hétéroatome de la fonction, quel que soit le type des liaisons (simple, double ou triple), sera ajouté à la fonction.

Cette règle permet, par exemple, d'obtenir des fonctions telles les acétals, les acides, les esters :



(2) Tout carbone lié à au moins deux hétéroatomes de la fonction, quel que soit le type des liaisons, devra être ajouté à la fonction.

Il s'agit bien entendu d'un complément obligatoire à la règle 1 (d'autant plus que les deux hétéroatomes n'ont pas forcément été obtenus par la règle 1 dans le cas de cycle).

(3) Tout carbone lié à un seul hétéroatome de la fonction sera ajouté à cette fonction si la liaison entre ces deux atomes est double ou triple.

Des fonctions telles les nitriles, les cétones, etc... qui n'ont pu être obtenues par les premières régles pourront l'être ici :

(4) Deux carbones liés entre eux par une liaison multiple seront ajoutés à la fonction si au moins l'un d'eux est relié à un hétéroatome de la fonction par une liaison simple.

(4') Un carbone lié par une liaison double à un carbone de la fonction, lui-même lié par une liaison double de la même fonction, sera ajouté à cette fonction. Seules les liaisons doubles sont ici envisageables, étant donné la tétravalence du carbone.

Ces deux dernières règles permettent la mise en évidence de fonctions comme les éthers d'énol ou les cétènes :

Cette liste de règles a été établie sur base d'observations des fonctions chimiques les plus courantes. L'exhaustivité de cette liste ne pourra cependant être établie qu'à la suite de nombreuses observations supplémentaires.

2.1.1.2 Liaisons multiples carbone-carbone

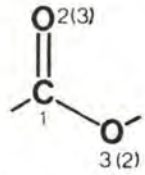
Il reste maintenant à rechercher les liaisons multiples (entre carbones) isolées c'est-à-dire celles qui ne sont pas reprises dans les fonctions précédemment reconnues.

Il faut faire, ici, un commentaire au sujet des doubles liaisons. En effet, ce type de liaison possède en général une géométrie définie par rapport à la "taille" des groupements qui y sont reliés. Elle est définie sur base, non seulement de la masse des atomes constitutifs des groupements, mais aussi de la multiplicité des liaisons entre ces atomes. Il va sans dire qu'une telle évaluation peut se révéler extrêment complexe d'autant plus qu'elle sera déterminée par les coordonnées d'un des atomes de chaque groupement sur la double liaison. Or, au stade où nous sommes, rappelons-nous que nous avons décidé de remplacer les coordonnées des atomes par un numéro. Il semble donc plus judicieux de laisser le soin de cette détermination au chimiste qui pourra, s'il le veut, qualifier chacune de ces liaisons.

2.1.2 Représentation des fonctions

Une fois les fonctions de la molécule isolées, il faut encore trouver une méthode pour les représenter de manière à la rendre facilement identifiable. Une fonction peut être vue, à ce stade, comme une molécule et alors être codée avec un algorithme comme celui de Morgan (il est présenté en annexe). Celui-ci présente cependant le problème d'établir une numérotation des atomes sans tenir compte des différences entre ceux-ci ni du type des liaisons entre atomes. Ce qui fait que si le "squelette" de la fonction présente une symétrie, l'algorithme conduira à plusieurs

numérotations possibles des atomes. Nous pouvons voir un exemple de ce problème avec la fonction ester :



Nous obtenons deux numérotations possibles conduisant aux deux codes suivants :

connexions : 1/1 types des atomes : C/O/O
atomes : 2 3 types des liaisons : 1/2

ou

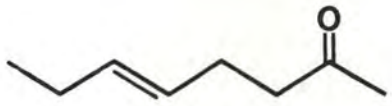
connexions : 1/1 types des atomes : C/O/O
atomes : 2 3 types des liaisons : 2/1

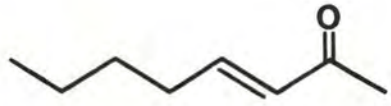
La solution consiste simplement à toujours choisir le même code chaque fois qu'une même fonction est détectée. Une façon de réaliser ce choix sera d' attribuer à chaque atome de chaque description son numéro atomique. En sommant alors ces valeurs dans l'ordre d'appariton dans chaque code, on retient celle dont la somme devient le plus rapidement la plus élevée. Si cela n'est pas suffisant, on procède de même avec la multiplicité des liaisons. Si les descriptions ne peuvent toujours pas être différenciées de la sorte, c'est que la fonction présente vraiment une symétrie (comme dans le cas d'un acétal, par exemple). En appliquant cette solution à notre exemple précédent, on retiendra le deuxième code. En effet, les atomes apparaissant dans le même ordre dans les deux codes, mais dans le deuxième, la double liaison est décrite avant la liaison simple.

Un algorithme de codage des fonctions n'a aucun intérêt dans le cas des liaisons multiples carbone-carbone isolées. Celles-ci font en effet toujours intervenir deux atomes de carbones identiques (par rapport à cette liaison). Il suffira simplement d'en préciser la multiplicité.

2.2 Description des liens entre fonctions

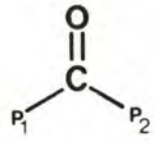
Lorsque des fonctions sont voisines, leurs réactivités peuvent s'influencer. Il serait donc intéressant de déterminer, dans certaines limites, ces relations de proximité. Il est par exemple important de pouvoir différencier les deux molécules suivantes :





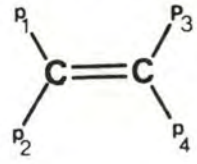
2.2.1 Prolongements des fonctions

On entend par "prolongement d'une fonction" les liaisons dont un atome appartient à la fonction, à l'inverse de l'autre. Cette définition permettra de mettre facilement en évidence ces prolongements. Bien entendu, plusieurs parmi ceux-ci peuvent avoir le même atome de la fonction comme origine. Dans ce cas, ces derniers devront être considérés comme identiques. Ce sera le cas, par exemple, pour les cétones :



Il n'y a en effet aucune raison de distinguer les prolongements P1 et P2.

Le cas des doubles liaisons est cependant plus délicat :



Puisque nous avons décidé de ne pas tenir compte de la

géométrie de la liaison au niveau du codage, nous pouvons considérer que les prolongements P1 et P2 sont identiques ainsi que P3 et P4. Mais la symétrie de la double liaison nous oblige de plus à regarder les paires de prolongements (P1, P2) et (p3, P4) comme identiques aussi.

2.2.2 Etablissement des relations entre fonctions

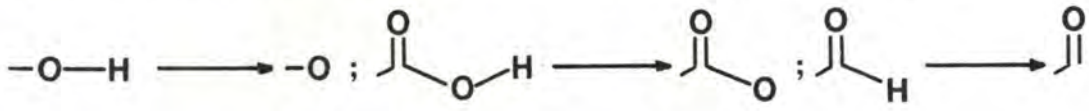
Pour ne pas rendre trop lourdes ces relations ni leur recherche, nous avons choisi de ne mettre en évidence que celles où le nombre d'atomes séparant deux fonctions n'excéderait pas un (donc deux liaisons). Une relation non précisée devra être considérée, lors d'une recherche, comme pouvant éventuellement exister mais pour laquelle les prolongements des fonctions en cause sont distants de plus de deux liaisons.

Du fait des règles de sélection des fonctions, les atomes séparant celles-ci ne pourront être que des carbones. Il ne sera donc pas nécessaire de les préciser. Il en sera de même pour les liaisons qui ne peuvent être que simples.

2.3 Remarque sur les hétéroatomes

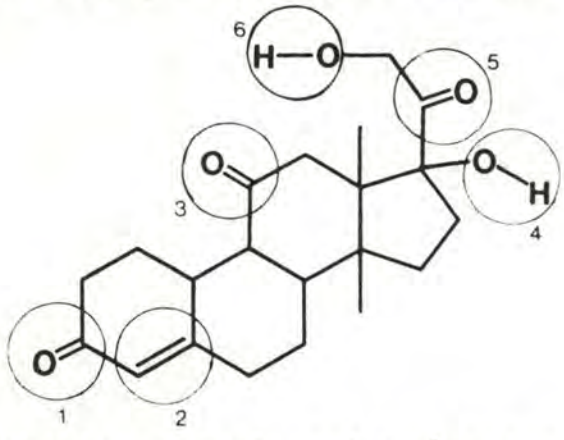
Nous venons de le voir, la description des fonctions est essentiellement basée sur les hétéroatomes. Or, tout symbole chimique introduit sur un sommet fera que ce sommet sera considéré comme un hétéroatome, *même si c'est un hydrogène ou un carbone*. Pour les carbones, le problème est simple : il suffit simplement d'en supprimer le symbole sur les sommets concernés. Dans le cas de l'hydrogène, ce n'est pas si évident. En effet, simplement supprimer le symbole reviendrait à remplacer un hydrogène par un carbone. Il faut

donc supprimer l'atome et la liaison qui le lie au reste de la molécule (cela ne pose aucun problème puisque l'hydrogène est monovalent et ne participe donc qu'à une seule liaison). Puisque cette opération modifie le squelette, elle devrait préférentiellement être effectuée avant sa description. On pourrait penser que cette suppression va entraîner un problème au niveau des fonctions comme les alcools, les acides, les aldéhydes etc... qui deviendraient respectivement des éthers, des esters, des cétones :



Il n'en sera cependant rien puisqu'à la description des fonctions s'ajoute celle des prolongements de cette fonction. Si le code obtenu par l'algorithme de Morgan est le même pour un acide et un ester, il n'en sera plus de même si on précise que l'acide possède un prolongement libre tandis que l'ester en possède deux.

Nous allons terminer en présentant un exemple pour lequel nous allons délimiter les fonctions telles qu'elles seront par la méthode que nous venons de décrire.



Les fonctions chimiques sont entourées sur le dessin. Le système établira une relation entre les fonctions 1 et 2 qui ne sont séparées par aucun carbone. On aura également un lien entre les fonction 4, 5 et 5, 6 où, cette fois, on observe une séparation de un carbone. Les autres ne seront pas relevées explicitement puisque trop distantes.

CHAPITRE IV : CONCLUSION

Nous venons de proposer un système permettant l'introduction et le codage de molécules dans un système informatique. Ces deux choses constituent le fondement de la création d'une base de données en chimie organique. Les molécules doivent cependant encore être regroupées pour définir les réactions chimiques. Celles-ci peuvent être représentées par le schéma suivant dans lequel les réactifs conduisent, après transformations aux produits.

REACTIF(S) $\xrightarrow{transformations}$ PRODUIT(S)

Cela ne devrait pas poser de problèmes difficiles à résoudre.

Il reste maintenant un point important à développer. C'est une méthode de recherche dans cet ensemble de molécules. Il est évident que lorsque le concept de réaction sera introduit, la caractérisation de chaque molécule en tant que réactif ou produit de réaction va déjà diviser à peu près par deux le nombre de molécules à explorer. C'est cependant nettement insuffisant car il est impensable d'appliquer les techniques de comparaison que nous avons développées sur un grand nombre de molécules. Il faudrait donc pouvoir sélectionner quelques molécules candidates sur lesquelles seront seulement effectués les tests. Les critères de sélection sont assez nombreux et peuvent porter sur le squelette et sur les fonctions. Parmi ceux-ci, nous pouvons relever

- le nombre de cycles de chaque taille
- le nombre de paires et de triplets de cycle
- l'existance d'un triplet caractéristique

- le nombre de chaînes
- le nombre de chaînes sur un cycle
- la présence d'une fonction
- ...

Une remarque s'impose en ce qui concerne plus particulièrement les critères en rapport avec le squelette, par exemple, le nombre de cycles. En effet, il ne faut pas nécessairement rechercher une molécule possédant exactement un nombre de cycles mais au moins ce nombre. C'est la même chose pour les triplets etc... On peut encore tirer d'autres constatations. Par exemple, une molécule qui possèderait le nombre de cycles exact mais un nombre supérieur de triplets par rapport à la molécule cible ne pourra sûrement pas être candidate. Cela signifierait effectivement que l'agencement de ces cycles est différent de celui de la molécule cible.

Il est probable que parmi ces critères, certains soient plus sélectifs que d'autres. Il s'agit donc là d'une proposition à étudier soigneusement mais qui devrait normalement conduire à de bons résultats.

En plus des recherches sur une molécule isolée, il sera possible d'en faire sur des transformations de molécules. Il faudra raisonner, ici, sur le concept de "réaction". La recherche portera donc sur deux ou plusieurs molécules (au moins un réactif et un produit) liées par une réaction chimique.

ANNEXE I : NOTIONS DE CHIMIE ORGANIQUE

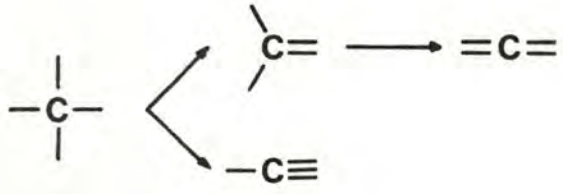
Etymologiquement, la chimie organique a pour objet l'étude des substances constituant les organismes vivants, d'où son nom. Par opposition, la chimie minérale étudie les composés minéraux que l'on trouve dans le sol et l'atmosphère. Cette définition de la chimie organique a cependant perdu son fondement philosophique puisque, maintenant, bien des composés "organiques" synthétisés en laboratoire n'ont jamais existés dans la nature. Il serait plus approprié d'appeler cette science la *chimie des composés du carbone*.

En effet, la base de tout produit organique est le carbone. Celui-ci possède une propriété assez exceptionnelle qui caractérise peu d'autres atomes : plusieurs atomes de carbones ont la possibilité de se lier entre eux pour former de longues chaînes. Le carbone étant un atome tétravalent, il peut prendre part à quatre liaisons chimiques et donc posséder quatre voisins. Cette condition ne sera cependant vérifiée que s'il s'agit de liaisons simples. Il y a, de fait, trois types de liaisons possibles pour le carbone :

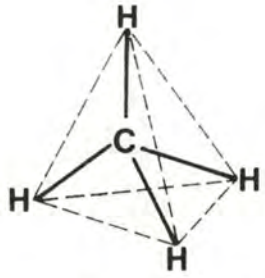
- les liaisons simples : C—C
- les liaisons doubles : C=C
- les liaisons triples : C≡C

chacun d'entre eux ayant des propriétés propres. Si un carbone prend part à une liaison multiple, le nombre de liaisons auxquelles il pourra encore participer en sera réduit d'autant.

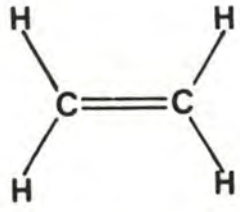
Ex. :



Un exemple simple de composé carboné est le méthane (un hydrocarbure) dont la formule développée dans l'espace est présentée ci-dessous :

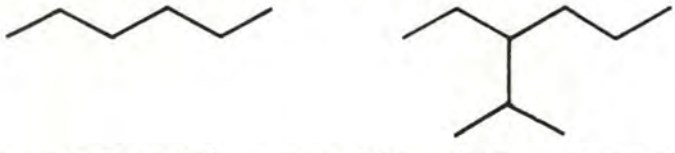


Cette molécule n'est pas plane et forme en fait un tétraèdre dont les hydrogènes occupent les sommets, le carbone étant au centre. La géométrie du carbone est dite ici SP_3. Dans d'autres cas, le carbone peut former des molécules planes. Ce sera le cas de l'éthylène où l'on observe alors une liaison double entre les carbones dont la géométrie est alors SP_2 :

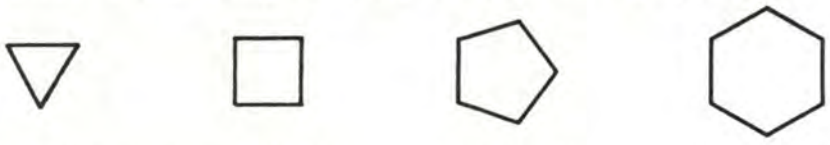


Dans l'exemple de l'acéthylène, nous avons des carbones SP :

Nous avons vu que le carbone peut former des chaînes. Celles-ci peuvent être ramifiées ou non :



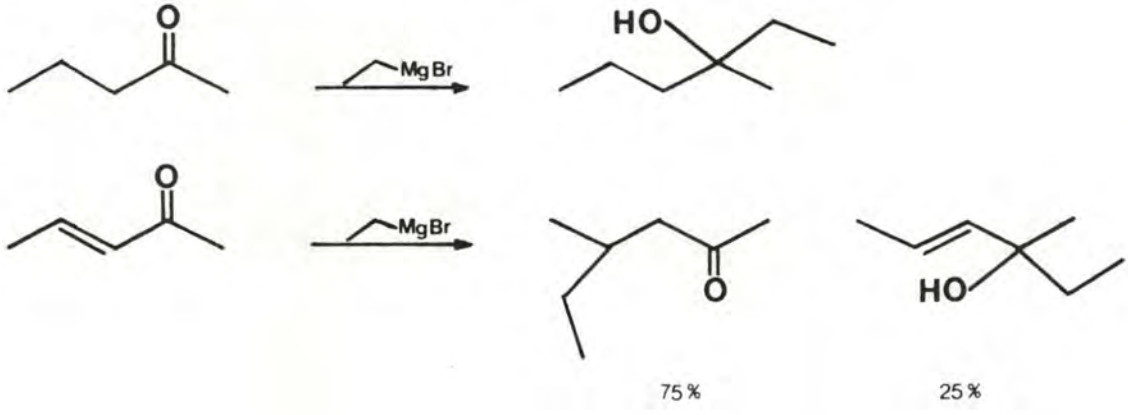
Si une chaîne est bouclée sur elle-même, on parlera alors de cycles. Il peut en exister de plusieurs tailles dont les plus courants sont composés de trois à six carbones.



cyclopropane cyclobutane cyclopentane cyclohexane

(Le cyclopropane est, actuellement, le plus puissant des anesthésiques par inhalation)

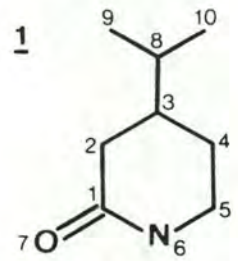
Outre le carbone, les composés organiques contiennent très souvent des atomes d'hydrogènes (comme les hydrocarbures, par exemple). On rencontre encore d'autres éléments, les hétéroatomes, comme l'oxygène (O) et l'azote (N), qui sont les plus importants après le carbone et l'hydrogène, mais aussi des non-métaux comme le chlore (Cl), le soufre (S), l'iode (I), etc... et des métaux comme le sodium (Na), le magnésium (Mg), etc... Tous ces éléments contribuent à la formation des fonctions chimiques qui constituent les entités réactionnelles des molécules. Si ces fonctions sont proches l'une de l'autre, leur réactivité peut-être considérablement modifiée. Dans l'exemple suivant, la cétone (C=O) des deux molécules pourra réagir très différemment et cela, à cause de la présence de la double liaison que l'on dira "conjuguée" à la cétone (du fait qu'elle n'en est séparée que par une liaison) :

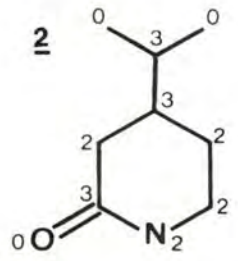


ANNEXE II : ALGORITHME DE MORGAN

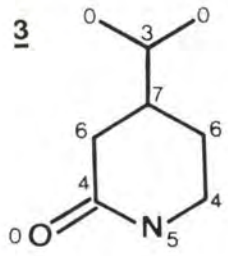
Cet algorithme [8] est normalement utilisé pour représenter une molécule complète. Le code obtenu reprend toutes les informations contenues dans le dessin d'une molécule et permet dès lors de la reconstruire sans problème. L'intérêt de ce code est qu'il peut facilement être manipulé dans un ordinateur car il est, comme nous allons le voir, représenté sous forme d'une chaîne de nombres et de caractères. Le principe de cet algorithme est de classer les atomes d'une molécule en fonction du nombre de leurs voisins et en cas d'égalité, en fonction du nombre de voisins de leurs voisins. Nous allons le présenter sur base de l'exemple proposé dans l'article de référence.

Les atomes sont d'abord numérotés (1) de façon arbitraire. On attribue ensuite à chaque atome, et indépendamment de la première numérotation, un nombre égal à celui de ses voisins (2). Ce nombre est appelé "connectivité de l'atome".



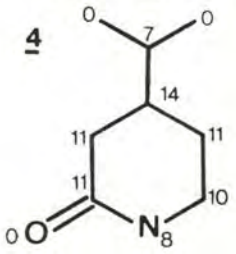


Remarquons que les atomes d'hydrogène ne sont pas considérés et qu'en plus, les atomes terminaux (reliés à un seul autre atome) ont reçu, et recevront toujours par convention, la valeur 0. On attribue ensuite à chacun des atomes un nombre égal à la somme des connectivités de ses voisins (3). C'est la connectivité étendue.

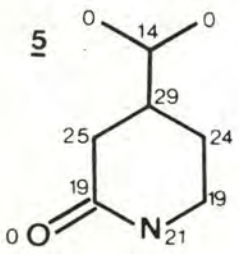


6 classes

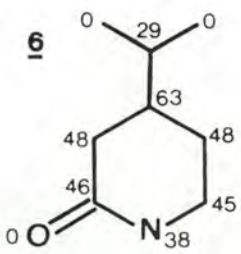
Sur cette base, on compte le nombre de classes, une classe regroupant tous les atomes de même connectivité. Nous obtenons 6 classes. Dans la figure 2, nous en aurions obtenu seulement 3. A partir de la figure 3, on recalcule les connectivités étendues (4, 5, 6 ,7) jusqu'à ce que le nombre de classes devienne inférieur à celui obtenu à l'itération précédente ou que le nombre de classes soit le même pour la troisième fois consécutive, ce que nous obtenons pour la figure 7.



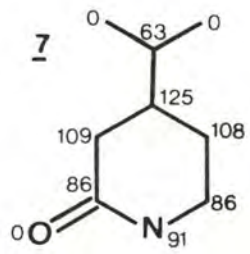
6 classes



7 classes

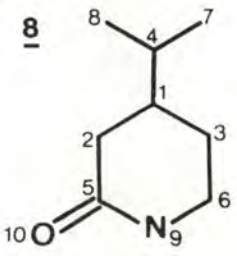


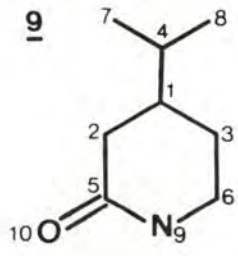
7 classes



7 classes

On établit ensuite, à partir des classes, une "numérotation canonique". On part, pour cela, d'un des atomes de la classe de connectivité étendue associée à la valeur la plus élevée, ici 125. On numérote ensuite les voisins de cet atome en choisissant d'abord ceux de plus grande connectivité et l'on répète l'opération jusqu'à ce que tous les atomes aient reçu un numéro. On obtient les figures 8 et 9.





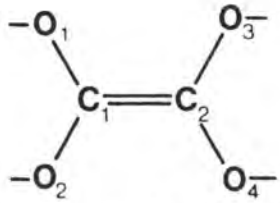
On obtient deux numérotations à cause des atomes 7 et 8 qu'il n'est pas possible de différencier. On peut maintenant construire la table de connexions canonique à partir d'une de ces numérotations :

```
connexions :  1/1/1/2/3/4/4/5/5  ;  6-9
    atomes :  2 3 4 5 6 7 8 9 10
types des atomes :  C/C/C/C/C/C/C/C/N/O
                    1/2/3/4/5/6/7/8/9/10
types des liaisons :  1/1/1/1/1/1/1/1/2/1
```

La liste de connections donne pour chaque atome, sauf le premier, le plus petit numéro canonique parmi ses voisins, et qui doit être inférieur à son propre numéro. Par exemple, l'atome 2 est lié au 1, le 3 au 1, etc... Toutes les liaisons de la molécule sont ainsi décrites, sauf la fermeture du cycle. C'est pourquoi la liaison 6-9 a dû être rajoutée séparément. Il reste maintenant à fournir les listes des types des atomes et des types de liaisons (1 = simple liaison, 2 = double liaison et 3 = triple liaison).

Lorsque nous avons discuté du codage des fonctions, nous avons déjà dit que l'algorithme ne tenait compte ni de la nature des liaisons ni de la nature des atomes pour leur attribuer un numéro. Nous avons donc proposé une méthode pour choisir une représentation particulière lorsqu'il s'en présente plusieurs. Le choix risque cependant de s'avérer impossible si la fonction présente vraiment une symétrie. Cette symétrie peut avoir des répercutions sur les prolongements. Si deux de ceux-ci sont identiques suite à une symétrie il ne s'agit pas de les différencier simplement parce qu'ils n'ont pas le même atome de la fonction comme origine. Admettons, par exemple, que l'on ait pu identifier

la fonction suivante :



Les numéros en indice sont attribués de façon arbitraire à chaque atome. ils serviront à les différencier. On peut obtenir 8 listes différentes d'atomes. Il n'est pas possible d'en choisir une préférentiellement à l'autre. La liste des liaisons est identique pour chacune d'elles.

<u>listes des types d'atomes</u> :

$C_1/C_2/O_1/O_2/O_3/O_4$	$C_2/C_1/O_3/O_4/O_1/O_2$
$C_1/C_2/O_2/O_1/O_3/O_4$	$C_2/C_1/O_4/O_3/O_1/O_2$
$C_1/C_2/O_1/O_2/O_4/O_3$	$C_2/C_1/O_3/O_4/O_2/O_1$
$C_1/C_2/O_2/O_1/O_4/O_3$	$C_2/C_1/O_4/O_3/O_2/O_1$

<u>liste des connexions</u> : 1/1/1/2/2
2/3/4/5/6

<u>liste des types de liaisons</u> : 2/1/1/1/1

Nous allons tout d'abord considérer les deux colonnes de listes des types d'atomes séparément. Nous voyons que chaque élément des paires (O_1,O_2) et (O_3,O_4) occupe indifféremment la place de l'un ou de l'autre. Si nous regardons la fonction, nous observons effectivement que les atomes O_1,O_2 occupent des positions identiques ainsi que les atomes O_3 et O_4. Nous allons maintenant comparer les deux colonnes. Cette fois, on voit que les atomes C_1 et C_2 sont identiques. Mais cela n'a pas beaucoup d'importance puisqu'ils ne sont origines d'aucun prolongement de la fonction. On peut aussi se rendre compte que les paires (O_1,O_2) et (O_3,O_4) sont identiques. Et elles sont origines de prolongements. Il faut donc s'en préoccuper. Il est donc important de relever tous les codes que l'on peut obtenir avec l'algorithme de Morgan pour pouvoir déterminer les prolongements identiques dans une fonction.

ANNEXE III : LISTING DES PROGRAMMES

Nous avons essentiellement développé un programme permettant l'introduction graphique des molécules. Il composé de deux parties :

- un programme assembleur pour la gestion du graphisme et de la souris. Les procédures sont facilement appelées par un programme écrit en C. Les méthodes d'interfaçage sont décrites dans le mode d'emploi du Lattice C.
- Le programme proprement dit, écrit en C, qui gère l'introduction graphique des molécules.

Nous avons également écrit un programme réalisant la description des cycles sous forme de triplets, avec recherche d'une liste minimale de triplets, et le codage des chaînes.

```
;
;Ce programme offre une série de procédures graphiques et de gestion de la
;souris qui peuvent facilement être exploitées par des programmes en C. Les
;procédures de gestion de la souris se contentent en fait de rendre accessibles
;les primitives offertes par le gestionnaire de souris MOUSE.
;Les primitives offertes sont les suivantes :
;         SETGRS ()                              gestion du graphisme
;         SETNORS ()
;         LINE (X1,Y1,X2,Y2,P1,P2)
;         HLINE (X1,X2,Y)
;         VLINE (X,Y1,Y2)
;         PRCARGR (X,Y,C)
;         MOUSINIT (numcurs)                     gestion de la souris
;         SELCURS (numcurs)
;         FORMACURS (C1,C2)
;         WINDCURS (Xmin,Xmax,Ymin,Ymax)
;         PRCLCURS (S)
;         GETPOSC (pos[2])
;         PUTPOSC (pos[2])
;                 ----------------------*---------------------
;
DGROUP    GROUP    DATA
DATA      SEGMENT  WORD PUBLIC 'DATA'
          ASSUME   DS:DGROUP
;
;Variables intermédiaires utilisées pour le tracé des droites
;
X2        DW       1 DUP (?)          ;contient la valeur de X2
Y2        DW       1 DUP (?)          ;contient la valeur de Y2
INVERS    DB       1 DUP (?)          ;1 si couples inversés, sinon 0
;
;Codes graphiques des représentations des lettres majuscules et minuscules
;sur 5 points en hauteur et 3 en largeur. Chaque lettre est représentée par
;5 bytes. Dans chaque byte, un point est représenté par deux bits. Le premier
;point est toujours à 0 pour l'espacement entre lettre. Les 3 derniers dans
;chaque byte représente donc un point de la lettre.
;Trois caractères sont définis par ligne, d'abord les majuscules, puis les
;minuscules.
;
CCARMAJ   DB       0CH,33H,33H,3FH,33H,3CH,33H,3CH,33H,3CH,0FH,30H,30H,30H,0FH
          DB       3CH,33H,33H,33H,3CH,3FH,30H,3CH,30H,3FH,3FH,30H,3CH,30H,30H
          DB       0FH,30H,30H,33H,0FH,33H,33H,3FH,33H,33H,3FH,0CH,0CH,0CH,3FH
          DB       3FH,03H,03H,33H,0CH,33H,3CH,30H,3CH,33H,30H,30H,30H,30H,3FH
          DB       33H,3FH,33H,33H,33H,03H,33H,3FH,33H,30H,0CH,33H,33H,33H,0CH
          DB       3FH,33H,3FH,30H,30H,3FH,33H,33H,3CH,03H,3FH,33H,3FH,3CH,33H
          DB       0FH,30H,0CH,03H,3CH,3FH,0CH,0CH,0CH,0CH,33H,33H,33H,33H,3FH
          DB       33H,33H,33H,33H,0CH,33H,33H,3FH,3FH,33H,33H,33H,0CH,33H,33H
          DB       33H,33H,0CH,0CH,0CH,3FH,03H,0CH,30H,3FH
;
CCARMIN   DB       00H,3CH,03H,3FH,3FH,30H,30H,3FH,33H,3FH,00H,0FH,30H,30H,0FH
          DB       03H,03H,3FH,33H,3FH,00H,0CH,3FH,30H,0FH,0FH,30H,3CH,30H,30H
          DB       00H,3FH,3FH,03H,3CH,30H,30H,3CH,33H,33H,00H,0CH,0CH,0CH,0CH
          DB       00H,03H,03H,33H,0CH,30H,30H,33H,3CH,33H,0CH,0CH,0CH,0CH,0FH
          DB       00H,33H,3FH,33H,33H,00H,0FH,33H,33H,33H,00H,0CH,33H,33H,0CH
          DB       00H,3FH,3FH,30H,30H,00H,3FH,3FH,03H,03H,00H,0FH,30H,30H,30H
          DB       00H,0FH,30H,03H,3CH,30H,3CH,30H,30H,0FH,00H,33H,33H,33H,3FH
          DB       00H,33H,33H,33H,0CH,00H,33H,33H,3FH,33H,00H,00H,33H,0CH,33H
          DB       00H,33H,33H,0CH,0CH,00H,3FH,03H,30H,3FH
;
CSPACE    DB       5 DUP (0)
;
```

```
;Définition du curseur de souris en forme de '+'.
;
MOTIF1    DW        16 DUP (0FFFFH)
          DW        5 DUP (0),300H,300H,3FF0H,300H,300H,6 DUP (0)
;
;Définition du curseur en forme de point.
;
MOTIF2    DW        16 DUP (0FFFFH)
          DW        6 DUP (0),3 DUP (0FC0H),7 DUP (0)
;
;Définition du curseur en forme de flèche.
;
MOTIF3    DW        16 DUP (0FFFFH)
          DW        0C000H,0F000H,0FC00H,0FF00H,0FFC0H,0FFF0H,0FC00H
          DW        0C300H,300H,0C0H,0C0H,30H,30H,0,0,0
;
;Définition du curseur en forme de double flèche horizontale.
;
MOTIF4    DW        16 DUP (0FFFFH)
          DW        5 DUP (0),0C30H,300CH,0C003H,300CH,0C30H,6 DUP (0)
;
;Définition du curseur en forme de double flèche verticale.
;
MOTIF5    DW        16 DUP (0FFFFH)
          DW        4 DUP (0),0C0H,330H,0C0CH,0,0,0C0CH,330H,0C0H,4 DUP (0)
;
;Zone réservée à la définition d'un curseur alphabétique.
;
MOTIF6    DW        4 DUP (0FFFFH)
DEBME     DW        7 DUP (?),5 DUP (0FFFFH)
          DW        5 DUP (0)
DEBMC     DW        5 DUP (?),6 DUP (0)
;
;Nombre de caractères du curseur alphabétique (1 ou 2)
;
NCARACT   DB        1 DUP (?)
;
DATA      ENDS
;
PGROUP    GROUP     PROG
PROG      SEGMENT   BYTE PUBLIC 'PROG'
          ASSUME    CS:PGROUP
;
;Tracé d'une droite entre deux points.
;  Syntaxe : LINE (X1,Y1,X2,Y2,C,P).
; 0 <= X1 et X2 <= 319
; 0 <= Y1 et Y2 <= 199
;Si ces conditions ne sont pas respectées, la droite n'est pas affichée.
;Si C = 1, on trace une droite modifiable.
;Si C = 2, on rend une droite modifiable définitive.
;Si C = 0, on efface une droite. Si elle était modifiable, les points sur
;lesquels elle a été tracée ne sont pas effacés.
;Si P est à 0, le point correpondant au couple (X1,Y1) ne sera pas affiché.
;
;Lorsque l'on trace une droite modifiable, les points qu'elle recouvre
;éventuellement seront plus lumineux. Lors de l'effacement de cette droite,
;les points recouverts retrouveront leur luminosité originale. Si la droite
;est rendue définitive, les points plus lumineux deviendront normaux. Si la
;droite est alors effacée, les points recouverts le seront également.
;
;Une même droite sera toujours imprimée dans le même sens, même si l'ordre des
;couples est inversé. On évite ainsi les petites différences qui pourraient se
;produire autrement.
;
```

```
        PUBLIC  LINE
LINE    PROC    NEAR
X       EQU     4               ;index première coordonnée dans la pile
        PUSH    BP
        MOV     BP,SP
        MOV     AX,[BP+X+8]     ;argument C
        CMP     AL,1            ;initialisation de SETPT en fonction de C
        JZ      ADJSP3
        CMP     AL,0
        MOV     AL,64
        MOV     BYTE PTR CS:SETP3+1,AL
        MOV     AL,191
        MOV     BYTE PTR CS:SETP4+1,AL
        MOV     AX,0E022H
        MOV     WORD PTR CS:SETP6,AX
        JZ      ADJSP1
        MOV     AL,255
        JMP     ADJSP2
ADJSP1: MOV     AL,63
ADJSP2: MOV     BYTE PTR CS:SETP5+1,AL
        JMP     ADJSP4
ADJSP3: MOV     AL,192
        MOV     BYTE PTR CS:SETP3+1,AL
        MOV     BYTE PTR CS:SETP4+1,AL
        MOV     AL,128
        MOV     BYTE PTR CS:SETP5+1,AL
        MOV     AX,0E00AH
        MOV     WORD PTR CS:SETP6,AX
ADJSP4: MOV     CX,[BP+X]       ;argument X1
        CMP     CX,320
        JC      ARG1
        JMP     RETURN          ;si erreur
ARG1:   MOV     DX,[BP+X+2]     ;argument Y1
        CMP     DX,200
        JC      ARG2
        JMP     RETURN          ;si erreur
ARG2:   MOV     AX,[BP+X+4]     ;argument X2
        CMP     AX,320
        JC      ARG3
        JMP     RETURN          ;si erreur
ARG3:   MOV     BX,[BP+X+6]     ;argument Y2
        CMP     BX,200
        JC      ARG4
        JMP     RETURN          ;si erreur
ARG4:   MOV     INVERS,0
        CMP     CX,AX
        JC      ARG5
        XCHG    AX,CX           ;pour toujours imprimer une même droite
        XCHG    BX,DX           ;dans le même sens, on inverse les couples
        MOV     INVERS,1        ;si nécessaire
        CALL    SETP            ;point (x2,y2) qui est devenu le premier
        JMP     ARG6
ARG5:   CMP     WORD PTR [BP+X+10],0  ;impression ou non du premier point
        JZ      ARG6                  ;quand la droite n'est pas inversée.
        CALL    SETP
ARG6:   MOV     X2,AX
        MOV     Y2,BX
        SUB     AX,CX           ;EX
        JNS     DROIT1
        NEG     AX              ;si CX > AX
DROIT1: SUB     BX,DX           ;EY
        JNS     DROIT2
        NEG     BX              ;si DX > BX
DROIT2: CMP     AX,BX
```

```
          JNZ       DROIT3
          OR        AX,AX
          JNZ       DROIT3
          JMP       RETURN              ;si la droite est un point
DROIT3:   JC        DROIT5
          MOV       DI,AX
          CMP       INVERS,0
          JZ        DROIT4                    ;si les couples ont été inversés et
          CMP       WORD PTR [BP+X+10],1      ;qu'il ne fallait pas afficher (x1,y1)
          JZ        DROIT4                    ;c'est maintenant le dernier point
          DEC       DI                        ;qu'il ne faut pas afficher
DROIT4:   JMP       EXSEY
DROIT5:   XCHG      AX,BX
          MOV       DI,AX
          CMP       INVERS,0
          JZ        DROIT6
          CMP       WORD PTR [BP+X+10],1      ;paramètre p
          JZ        DROIT6
          DEC       DI                        ;pour ne pas imprimer le dernier point
DROIT6:   JMP       EYSEX
;
;Traitement du cas où EX > EY.
;    INPUT : AX = EX     ( ABS (X1 - X2) )
;            BX = EY     ( ABS (Y1 - Y2) )
;            CX = X1
;            DX = Y1
;            DI = nombre de points à afficher
;
;   Algorithme :
; Conditions initiales : dx := abs (x1-x2)
;                        dy := abs (y1-y2)
;                        x := x1; y := y1
;                        x1 < x2 (strictement) ==> dx <> 0
;    afficher le point (x,y)      * déjà fait si c'était nécessaire *
;    c := dx  (nombre de points à afficher)
;    p := dx/2
;A:  x := x+1
;    p := p+dy
;    si  p < dx  alors  afficher le point (x,y)
;                sinon  p := p-dx
;                       y := y+1
;                       afficher le point (x,y)
;    c := c-1
;    si  c <> 0  alors  aller en A
;
EXSEY:    PUSH      AX
          MOV       AL,4AH                    ;code de : DEC  DX
          CMP       DX,Y2
          JNC       EXSEY0
          MOV       AL,42H                    ;code de : INC  DX
EXSEY0:   MOV       BYTE PTR CS:EXSEY2,AL
          MOV       AL,49H                    ;code de : DEC  CX
          CMP       CX,X2
          JNC       EXSEY00
          MOV       AL,41H                    ;code de : INC  CX
EXSEY00:  MOV       BYTE PTR CS:EXSEY1,AL
          POP       AX
          MOV       BP,AX
          SHR       AX,1              ;EX/2
EXSEY1:   INC       CX
          ADD       AX,BX
          CMP       AX,BP
          JC        EXSEY3
          SUB       AX,BP
```

```
EXSEY2:   INC     DX                ;ou DEC  DX
EXSEY3:   CALL    SETP
          DEC     DI
          JNZ     EXSEY1
          JMP     RETURN
;
;Traitement du cas où EY > EX.
;    INPUT : AX = EY
;            BX = EX
;            DX = Y1
;            CX = X1
;            DI = nombre de points à afficher
;Algorithme et conditions initiales similaires à EXSEY.
;
EYSEX:    PUSH    AX
          MOV     AL,4AH
          CMP     DX,Y2
          JNC     EYSEX0
          MOV     AL,42H
EYSEX0:   MOV     BYTE PTR CS:EYSEX1,AL
          MOV     AL,49H
          CMP     CX,X2
          JNC     EYSEX00
          MOV     AL,41H
EYSEX00:  MOV     BYTE PTR CS:EYSEX2,AL
          POP     AX
          MOV     BP,AX
          SHR     AX,1
EYSEX1:   INC     DX
          ADD     AX,BX
          CMP     AX,BP
          JC      EYSEX3
          SUB     AX,BP
EYSEX2:   INC     CX
EYSEX3:   CALL    SETP
          DEC     DI
          JNZ     EYSEX1
          JMP     RETURN
;
RETURN:   POP     BP
          RET
LINE      ENDP
;
;Modifie un point à l'écran dans les coordonnées 320 x 200.
;Si ce point était déjà allumé, la couleur 3 est choisie, sinon,c'est la 2.
;Les coordonnées du point doivent être dans
;   CX pour X
;   DX pour Y
;Les valeurs de CX et DX sont supposées correctes.
;Aucun registre n'est modifié (sauf F).
;
;Initialisation pour une droite de construction :
;       SETP3 + 1   :  192
;       SETP4 + 1   :  192
;       SETP5 + 1   :  128
;       SETP6       :  0AH, E0H  (codes de OR  AH,AL)
;
;Initialisation pour rendre une droite définitive :
;       SETP3 + 1   :  64
;       SETP4 + 1   :  191
;       SETP5 + 1   :  255
;       SETP6       :  22H, E0H  (codes de AND  AH,AL)
;
```

```
;Initialisation pour effacer une droite :
;       SETP3 + 1   :  64
;       SETP4 + 1   :  191
;       SETP5 + 1   :  63
;       SETP6       :  22H, E0H  (codes de AND  AH,AL)
;
SETP      PROC      NEAR
          PUSH      DX
          PUSH      CX
          PUSH      BX
          PUSH      AX
          PUSH      DS
          SHR       DX,1
          JC        SETP1
          MOV       AX,0B800H          ;si ligne paire
          JMP       SETP2
SETP1:    MOV       AX,0BA00H          ;si ligne impaire
SETP2:    MOV       DS,AX
          MOV       AX,80
          MUL       DX
          MOV       BX,AX
          XOR       AL,AL
          SHR       CX,1               ;divise CX par 4 et AL sera utilisé
          RCR       AL,1               ;pour déterminerla position du point
          SHR       CX,1               ;dans le byte
          RCR       AL,1
          ADD       BX,CX              ;offset du byte contenant le point
          MOV       AH,[BX]
          MOV       CL,5
          SHR       AL,CL              ;position du point dans le byte (0,2,4,6)
          MOV       CL,AL              ;(0 si le pt est sur les bits 7 et 6 ...)
          ROL       AH,CL              ;place les 2 bits du pt ds les bits 6,7
SETP3:    MOV       AL,192             ;test de ces bits
          AND       AL,AH
          JZ        SETP5
SETP4:    MOV       AL,192
          JMP       SETP6
SETP5:    MOV       AL,128
SETP6:    OR        AH,AL              ;modifie état du point
          ROR       AH,CL              ;on remet le point à sa place
SETP8:    MOV       [BX],AH
          POP       DS
          POP       AX
          POP       BX
          POP       CX
          POP       DX
          RET
SETP      ENDP
;
;Cette procédure permet de tracer rapidement des droites verticales.
;Il suffit en effet de calculer l'adresse du premier point. Pour celle des
;points suivant, il suffit d'ajouter chaque fois 80 et la position du point
;dans le byte est toujours la même. Cela doit bien sûr être fait dans les
;deux segments B800H et BA00H.
;  Syntaxe : VLINE (X, Y1, Y2)
;Une seule valeur de X est nécessaire et Y1 et Y2 délimitent la droite. Les
;valeurs doivent respecter les limites de l'écran, sinon la droite n'est
;pas tracée.
;
          PUBLIC    VLINE
VLINE     PROC      NEAR
          PUSH      BP
          MOV       BP,SP
          MOV       DX,[BP+X+2]          ;paramètre Y1
```

```
        CMP     DX,200
        JNC     VLINEER
        MOV     CX,[BP+X+4]         ;paramètre Y2
        CMP     CX,200
        JNC     VLINEER
        CMP     CX,DX
        JNC     VLINE1
        XCHG    CX,DX
VLINE1: SUB     CX,DX
        INC     CX                  ;nombre de points à imprimer
        MOV     SI,[BP+X]           ;paramètre X
        CMP     SI,320
        JNC     VLINEER
        SHR     DX,1
        JC      VLINE2              ;si on commence par une ligne impaire
        MOV     WORD PTR CS:VLINE4+1,0B800H
        MOV     WORD PTR CS:VLINE7+1,0BA00H
        MOV     BX,OFFSET CS:VLINE9
        JMP     VLINE3
VLINEER: JMP    VLINE12
VLINE2: MOV     WORD PTR CS:VLINE4+1,0BA00H
        MOV     WORD PTR CS:VLINE7+1,0B800H
        MOV     BX,OFFSET CS:VLINE10
VLINE3: MOV     AX,80               ;comme pour SETP
        MUL     DX
        MOV     DX,AX
        XOR     AL,AL
        SHR     SI,1
        RCR     AL,1
        SHR     SI,1
        RCR     AL,1
        ADD     SI,DX               ;adresse premier point
        PUSH    CX
        MOV     CL,5
        SHR     AL,CL
        MOV     CL,AL
        MOV     AL,128              ;dètermine la couleur des points
        ROR     AL,CL
        POP     CX
        PUSH    ES
VLINE4: MOV     DX,0
        MOV     ES,DX
        PUSH    SI
        SHR     CX,1
        PUSH    CX
        JNC     VLINE5
        OR      ES:[SI],AL          ;si le nombre de points est impair
        ADD     SI,80
VLINE5: CMP     CX,0
        JZ      VLINE11             ;si la ligne est un point
VLINE6: OR      ES:[SI],AL          ;écrit les pts dans le premier segment
        ADD     SI,80
        LOOP    VLINE6
        POP     CX
        POP     SI
VLINE7: MOV     DX,0
        MOV     ES,DX
VLINE8: JMP     BX
VLINE9: OR      ES:[SI],AL          ;écrit les points dans le deuxième
        ADD     SI,80               ;segment si c'est 0BA00H
        LOOP    VLINE9
        JMP     VLINE11
VLINE10: ADD    SI,80               ;écrit les points dans le premier
        OR      ES:[SI],AL          ;segment si c'est 0B800H
```

```
        LOOP    VLINE10
VLINE11:POP     ES
VLINE12:POP     BP
        RET
VLINE   ENDP
;
;Cette procédure permet de tracer rapidement une droite horizontale.
;Une fois l'adresse du premier point calculée, les points suivants sont
;facilement atteints par décalage à l'intérieur du même byte et par
;incrémentation de 1 de l'adresse pour passer au byte suivant.
;  Syntaxe HLINE (X1, X2, Y)
;X1 et X2 représentent les limites de la ligne. Les valeurs doivent respecter
;les limites de l'écran sinon, la droite n'est pas tracée.
;
        PUBLIC  HLINE
HLINE   PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        MOV     DX,[BP+X]           ;paramètre X1
        CMP     DX,320
        JNC     HLINE7
        MOV     CX,[BP+X+2]         ;paramètre X2
        CMP     CX,320
        JNC     HLINE7
        CMP     CX,DX
        JNC     HLINE1
        XCHG    CX,DX
HLINE1: SUB     CX,DX
        INC     CX
        MOV     BX,[BP+X+4]         ;paramètre Y
        CMP     BX,200
        JNC     HLINE7
        PUSH    ES
        SHR     BX,1
        JC      HLINE2              ;détermine dans quel segment on est
        MOV     AX,0B800H
        JMP     HLINE3
HLINE2: MOV     AX,0BA00H
HLINE3: MOV     ES,AX
        MOV     AX,80
        PUSH    DX
        MUL     BX
        POP     DX
        MOV     BX,AX
        XOR     AL,AL               ;même calcul que SETP
        SHR     DX,1
        RCR     AL,1
        SHR     DX,1
        RCR     AL,1
        ADD     BX,DX               ;adresse du premier point
        PUSH    CX
        MOV     CL,5
        SHR     AL,CL
        MOV     CL,AL
        MOV     AL,128              ;sélection de la couleur
        MOV     AH,AL
        ROR     AL,CL               ;premier point
        POP     CX
        XOR     DL,DL
HLINE4: OR      DL,AL
        ROR     AL,1
        ROR     AL,1
        CMP     AL,AH
        JZ      HLINE5
```

```
        LOOP    HLINE4
        OR      ES:[BX],DL
        JMP     HLINE6
HLINE5: OR      ES:[BX],DL
        XOR     DL,DL
        INC     BX                      ;passe au byte suivant
        LOOP    HLINE4
HLINE6: POP     ES
HLINE7: POP     BP
        RET
HLINE   ENDP
;
;Affichage d'un caractère centré sur un point.
;  Syntaxe : PRCARGR (X,Y,C)
;X et Y sont les coordonnées graphiques du point et C le code ASCII du
;caractère. Ce code doit représenter une lettre majuscule ou minuscule
;ou un espace.
;Il faut    2 < X < 317   et
;           2 < Y < 197
;Si ces conditions ne sont pas respectées, le caractère n'est pas affiché.
;
        PUBLIC  PRCARGR
PRCARGR PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        MOV     AX,[BP+X+4]             ;code du caractère
        CMP     AX,32
        JZ      PRCAR0                  ;si c'est un espace
        CMP     AX,65
        JC      PRCARER
        CMP     AX,91
        JC      PRCAR1
        CMP     AX,97
        JC      PRCARER
        CMP     AX,123
        JNC     PRCARER
        SUB     AX,97                   ;si C est une minuscule
        MOV     DL,5
        MUL     DL
        ADD     AX,OFFSET CCARMIN
        MOV     SI,AX
        JMP     PRCAR2
PRCAR0: MOV     SI,OFFSET CSPACE
        JMP     PRCAR2
PRCAR1: SUB     AX,65                   ;si C est une majuscule
        MOV     DL,5
        MUL     DL
        ADD     AX,OFFSET CCARMAJ
        MOV     SI,AX
        JMP     PRCAR2
PRCARER: JMP    PRCARF
PRCAR2: MOV     CX,[BP+X]               ;extraction du paramètre X
        CMP     CX,317
        JNC     PRCARER
        CMP     CX,3
        JC      PRCARER
        MOV     DX,[BP+X+2]             ;extraction du paramètre Y
        CMP     DX,197
        JNC     PRCARER
        CMP     DX,3
        JC      PRCARER
        SUB     CX,2
        SHR     DX,1
        JC      PRCAR3
```

```
        MOV     WORD PTR CS:PRCAR5+1,0B800H
        MOV     WORD PTR CS:PRCAR7+1,320
        MOV     WORD PTR CS:PRCAR8+1,0BA00H
        JMP     PRCAR4
PRCAR3: MOV     WORD PTR CS:PRCAR5+1,0BA00H
        MOV     WORD PTR CS:PRCAR7+1,240
        MOV     WORD PTR CS:PRCAR8+1,0B800H
PRCAR4: DEC     DX
        PUSH    ES
PRCAR5: MOV     AX,0
        MOV     ES,AX
        MOV     AX,80
        MUL     DX
        MOV     BX,AX
        XOR     AL,AL
        SHR     CX,1
        RCR     AL,1
        SHR     CX,1
        RCR     AL,1
        ADD     BX,CX
        MOV     CL,5
        SHR     AL,CL
        MOV     CL,AL
        PUSH    SI
        MOV     DL,3
PRCAR6: MOV     AX,3F00H            ;on efface le fond
        ROR     AX,CL
        AND     ES:[BX],AX
        MOV     AL,[SI]
        AND     AL,170              ;séléction de la couleur du caractère
        MOV     AH,0                ;on écrit les lignes graphiques
        ROR     AX,CL               ;de numéro impair composant
        OR      ES:[BX],AX          ;le caractère
        INC     SI
        INC     SI
        ADD     BX,80
        DEC     DL
        JNZ     PRCAR6
        POP     SI
        INC     SI
PRCAR7: MOV     AX,0
        SUB     BX,AX
PRCAR8: MOV     AX,0
        MOV     ES,AX
        MOV     AX,3F00H            ;on écrit une ligne vide au-dessus
        ROR     AX,CL               ;du caractère
        AND     ES:[BX],AX
        MOV     DL,2
PRCAR9: ADD     BX,80               ;on écrit les lignes de numéro impair
        MOV     AX,3F00H
        ROR     AX,CL
        AND     ES:[BX],AX          ;efface le fond
        MOV     AL,[SI]
        AND     AL,170              ;séléction couleur
        MOV     AH,0
        ROR     AX,CL
        OR      ES:[BX],AX
        INC     SI
        INC     SI
        DEC     DL
        JNZ     PRCAR9
        ADD     BX,80               ;on écrit une ligne vide après le
        MOV     AX,3F00H            ;caractère
        ROR     AX,CL
```

```
          AND       ES:[BX],AX
          POP       ES
PRCARF:   POP       BP
          RET
PRCARGR   ENDP
;
;Passage au mode graphique 320 x 200 points.
;  Syntaxe SETGRS ()
;
          PUBLIC    SETGRS
SETGRS    PROC      NEAR
          MOV       AH,0
          MOV       AL,4
          STI
          INT       16
          RET
SETGRS    ENDP
;
;Retour au mode 80 x 24 caractères.
;  Syntaxe SETNORS ()
;
          PUBLIC    SETNORS
SETNORS   PROC      NEAR
          MOV       AH,0
          MOV       AL,2
          STI
          INT       16
          RET
SETNORS   ENDP
;
;Initialisation de la souris et sélection du curseur.
;  Syntaxe : MOUSINIT (numcurs)
;La valeur numcurs est un numéro de motif de curseur.
;
          PUBLIC    MOUSINIT
MOUSINIT  PROC      NEAR
          MOV       AX,0
          STI
          INT       51
          JMP       SELCURS
MOUSINIT  ENDP
;
;Sélection du curseur désiré.
;  Syntaxe : SELCURS (numcurs)
;La valeur numcurs est un numéro de motif de curseur.
;
          PUBLIC    SELCURS
SELCURS   PROC      NEAR
          PUSH      BP
          PUSH      ES
          MOV       AX,DS
          MOV       ES,AX
          MOV       BP,SP
          MOV       AX,[BP+X+2]
          CMP       AX,1
          JZ        SCURS1
          CMP       AX,2
          JZ        SCURS2
          CMP       AX,3
          JZ        SCURS3
          CMP       AX,4
          JZ        SCURS4
          CMP       AX,5
          JZ        SCURS5
```

```
        JMP     SCURSR
SCURS1: MOV     BX,7
        MOV     CX,7
        MOV     DX,OFFSET MOTIF1
        JMP     SCURSF
SCURS2: MOV     BX,7
        MOV     CX,7
        MOV     DX,OFFSET MOTIF2
        JMP     SCURSF
SCURS3: MOV     BX,0
        MOV     CX,0
        MOV     DX,OFFSET MOTIF3
        JMP     SCURSF
SCURS4: MOV     BX,7
        MOV     CX,7
        MOV     DX,OFFSET MOTIF4
        JMP     SCURSF
SCURS5: MOV     BX,7
        MOV     CX,7
        MOV     DX,OFFSET MOTIF5
        JMP     SCURSF
SCURSF: MOV     AX,9
        STI
        INT     51
SCURSR: POP     ES
        POP     BP
        RET
SELCURS ENDP
;
;Formation d'un curseur alphabétique de deux lettres maximum.
;  Syntaxe FORMACUR (C1,C2).
;C1 et C2 doivent être des codes ASCII de lettres majuscules ou minuscules.
;Si le cursuer ne doit être formé que d'une lettre, C2 doit être à 0.
;
        PUBLIC  FORMACUR
FORMACUR PROC   NEAR
        PUSH    BP
        MOV     BP,SP
        MOV     AX,[BP+X]           ;premier paramètre
        CMP     AX,65
        JC      ERROR
        CMP     AX,91
        JC      MAJ1
        CMP     AX,97
        JC      ERROR
        CMP     AX,123
        JNC     ERROR
        SUB     AX,97               ;si C1 est une minuscule
        MOV     DL,5
        MUL     DL
        ADD     AX,OFFSET CCARMIN
        MOV     SI,AX
        JMP     CAR2
MAJ1:   SUB     AX,65               ;si C1 est une majuscule
        MOV     DL,5
        MUL     DL
        ADD     AX,OFFSET CCARMAJ
        MOV     SI,AX
        JMP     CAR2
ERROR:  JMP     FIN
CAR2:   MOV     AX,[BP+X+2]         ;deuxième paramètre
        CMP     AX,0
        JZ      NOCAR               ;si un seul caractère
        MOV     NCARACT,2
```

```
        CMP     AX,65
        JC      ERROR
        CMP     AX,91
        JC      MAJ2
        CMP     AX,97
        JC      ERROR
        CMP     AX,123
        JNC     ERROR
        SUB     AX,97                   ;si C2 est une minuscule
        MOV     DL,5
        MUL     DL
        ADD     AX,OFFSET CCARMIN
        MOV     DI,AX
        JMP     SETME
NOCAR:  MOV     DI,OFFSET CSPACE
        MOV     NCARACT,1
        JMP     SETME2
MAJ2:   SUB     AX,65                   ;si C2 est une majuscule
        MOV     DL,5
        MUL     DL
        ADD     AX,OFFSET CCARMAJ
        MOV     DI,AX
SETME:  MOV     AX,0                    ;fixe le masque écran si 2 caractères
        MOV     BX,OFFSET DEBME
        MOV     CX,7
SETME1: MOV     [BX],AX
        INC     BX
        INC     BX
        LOOP    SETME1
        JMP     SETMC
SETME2: MOV     AX,63                   ;fixe le masque écran si 1 caractère
        MOV     BX,OFFSET DEBME
        MOV     CX,7
SETME3: MOV     [BX],AX
        INC     BX
        INC     BX
        LOOP    SETME3
SETMC:  MOV     BX,OFFSET DEBMC         ;fixe le masque curseur
        MOV     CX,5
SETMC1: MOV     AH,[SI]
        MOV     AL,[DI]
        MOV     [BX],AX
        INC     SI
        INC     DI
        INC     BX
        INC     BX
        LOOP    SETMC1
        PUSH    ES                      ;changement du curseur
        MOV     AX,DS
        MOV     ES,AX
        MOV     DX,OFFSET MOTIF6
        MOV     AX,9
        MOV     CX,8
        CMP     NCARACT,1
        JZ      FIXC1
        MOV     BX,7
        JMP     FIXC2
FIXC1:  MOV     BX,3
FIXC2:  STI
        INT     51
        POP     ES
FIN:    POP     BP
        RET
FORMACUR ENDP
```

```
;
;Sélection de la fenêtre pour le déplacement de la souris.
;  Syntaxe : WINDCURS (Xmin, Xmax, Ymin, Ymax)
;Il faut Xmin < Xmax,
;        Ymin < Ymax,
;        Xmin et Xmax compris entre 0 et 319 et
;        Ymin et Ymax compris entre 0 et 199.
;Si les deux dernières conditions ne sont pas vérifiées, les ajsutements
;suivants peuvent être exécutés : 0 pour Xmin et Ymin, 319 pour Xmax et 199
;pour Ymax.
;
;
          PUBLIC   WINDCURS
WINDCURS  PROC     NEAR
          PUSH     BP
          MOV      BP,SP
          MOV      CX,[BP+X]          ;valeur Xmin
          CMP      CX,320
          JC       WINDC1
          MOV      CX,0               ;on considère que Xmin était < 0
WINDC1:   SHL      CX,1
          MOV      DX,[BP+X+2]        ;valeur de Xmax
          CMP      DX,320
          JC       WINDC2
          MOV      DX,319             ;on considère que Xmax était > 319
WINDC2:   SHL      DX,1
          STI
          MOV      AX,7
          INT      51
          MOV      CX,[BP+X+4]        ;valeur de Ymin
          CMP      CX,200
          JC       WINDC3
          MOV      CX,0               ;on considère que Ymin était < 0
WINDC3:   MOV      DX,[BP+X+6]        ;valeur de Ymax
          CMP      DX,200
          JC       WINDC4
          MOV      DX,199             ;on considère que Ymax était > 199
WINDC4:   MOV      AX,8
          INT      51
          POP      BP
          RET
WINDCURS  ENDP
;
;Affichage/effacement du curseur graphique.
;  Syntaxe : PRCLCURS (switch)
;Si switch = 1, le curseur est affiché et il est effacé si switch = 0.
;
          PUBLIC   PRCLCURS
PRCLCURS  PROC     NEAR
          PUSH     BP
          MOV      BP,SP
          MOV      AX,[BP+X]
          XOR      AX,1
          INC      AX
          CMP      AX,3
          JNC      PRCLC1
          STI
          INT      51
PRCLC1:   POP      BP
          RET
PRCLCURS  ENDP
;
```

```
;Recherche de la position de la souris et de l'état des interrupteurs.
;  Syntaxe : GETPOSC (position)
;La position est retournée dans le tableau position représentant le
;couple (X, Y) avec 0 <= X <= 319 et 0 <= Y <= 199.
;L'état des interrupteurs est la valeur de la fonction :
;    0 si aucun interrupteur enfoncé
;    1 si interrupteur gauche enfoncé
;    2 si interrupteur droit enfoncé
;    3 si les deux interrupteurs sont enfoncés
;
          PUBLIC   GETPOSC
GETPOSC   PROC     NEAR
          MOV      AX,3
          STI
          INT      51
          MOV      AX,BX
          SHR      CX,1
          PUSH     BP
          MOV      BP,SP
          MOV      BX,[BP+X]
          MOV      [BX],CX
          MOV      [BX+2],DX
          POP      BP
          RET
GETPOSC   ENDP
;
;Cette procédure fixe la position du curseur graphique.
;  Syntaxe : PUTPOSC (position)
;Le couple (X, Y) est représenté par le tableau position.
;Si les coordonnées ne respectent pas les dimensions de l'écran, la position
;du curseur est inchangée.
;
          PUBLIC   PUTPOSC
PUTPOSC   PROC     NEAR
          PUSH     BP
          MOV      BP,SP
          MOV      BX,[BP+X]
          MOV      CX,[BX]
          MOV      DX,[BX+2]
          SHL      CX,1
          CMP      CX,640
          JNC      PUTPC1
          CMP      DX,200
          JNC      PUTPC1
          MOV      AX,4
          INT      51
PUTPC1:   POP      BP
          RET
PUTPOSC   ENDP
;
PROG      ENDS
          END
```

```
#include <fcntl.h>

#define ll 14                   /* longueur liaison */

#define dlm 4                   /* angle et écart des liaisons multiples */
#define anglml 45

#define prochval ll*ll/4        /* définit 2 sommets proches */
#define adjval 17               /* définit si 2 cycles se superposent */
#define maxdef ll*ll/4          /* définit si un cycle peut être déformé */

#define dimalpha 22              /* int (ll + sqrt(prochval) + 1.5) */

#define cxmax 319                /* limites écran */
#define cymax 199

#define maxsom 1000              /* place max pour les liaisons simples */
#define maxcycle 1000            /* place max pour les cycles */
#define maxlm 1000               /* place max pour les liaisons multiples */
#define maxatom 1000             /* place max pour les symboles chimiques */
#define tmcycle 10               /* taille max d'un cycle */
#define dtmcycle 20              /* 2 x tmcycle */


int atome [maxatom];
int atomsuiv = 0;

int lmsommet [maxlm];
int lmsuiv = 0;

int sommet [maxsom];
int somsuiv = 0;

int cycle [maxcycle];
int cyclsuiv = 0;


int vlsin [360];
int vlcos [360];

int alpha [dimalpha] [dimalpha];



main ()
 {
  int pos [2];
  int n1,n2,n3,n4;
  int i,n,ecart;
  int angle,anglei;

  initalpha ();
  initvlsc ();
  setgrs ();
  mousinit (1);
  prclcurs (1);

  n1 = 0;
  while (n1 != 6)
   {
    if ((getposc (pos) == 3) || (n1 == 0))
      select (&n1,&n2,&n3,&n4);
    prclcurs (0);
    windcurs (1,cxmax-1,1,cymax-1);
```

```
    prclcurs (1);
    switch (n1)

     {
      case 1 :
        while (getposc (pos) != 0);
        while (getposc (pos) == 0);
        plcycle (n2,n4);
        break;
      case 2 :
        while (getposc (pos) != 0);
        while (getposc (pos) == 0);
        plsliais ();
        break;
      case 3 :
        while (getposc (pos) != 0);
        while (getposc (pos) == 0);
        plmliais ();
        break;
      case 4 :
        while (getposc (pos) != 0);
        platome (n2,n3);
        break;
      case 5 :
        while (getposc (pos) != 0);
        translat ();
        break;
      case 6 :
        setnors ();
        savemolec ();
        break;
     }
   }
 }


/*************************************************************************/
/* Sélection d'un motif à dessiner.                                      */
/*  MOTIF              VALEUR DE n1 VALEUR DE n2     VALEUR DE n3 VALEUR DE n4 */
/*  cycles                  1       taille du cycle  quelconque   1 : deform  */
/*                                                                0 : sinon   */
/*  liaisons simples        2       quelconque       quelconque   quelconque  */
/*  liaisons multiples      3       quelconque       quelconque   quelconque  */
/*  atomes                  4       caractère 1      caractère 2  quelconque  */
/*  translation             5       quelconque       quelconque   quelconque  */
/*  terminer                6       quelconque       quelconque   quelconque  */
/* Attention, cette procédure est directement liée à la définition de        */
/* l'écran.                                                                  */
/*************************************************************************/

select (pn1,pn2,pn3,pn4)
int *pn1,*pn2,*pn3,*pn4;
 {
  int pos [2];

  prclcurs (0);
  setgrs ();
  windcurs (0,cxmax,0,cymax);
  selcurs (3);
  printf ("\n \n \n   CYCLES    3    4    5    6    7    8");
  printf ("\n \n \n \n   LIAISONS SIMPLES");
  printf ("\n \n \n \n   LIAISONS MULTIPLES");
  printf ("\n \n \n \n   ATOMES");
```

```
 printf ("\n \n \n \n   TRANSLATION");
 printf ("\n \n \n \n   TERMINER");
 writegr (77,28,"DEUX\0");
 prclcurs (1);
 *pn4 = 0;
 *pn2 = 0;
 *pn1 = 0;
 while (*pn1 == 0)
  {
   while (getposc (pos) != 0);
   while (getposc (pos) == 0);
   if (pos[1] > 15)
     pos[1] = pos[1] - 16;
   *pn1 = pos[1] / 32 + 1;
   if (*pn1 > 6)
     *pn1 = 6;
   if (*pn1 == 1)
    {
     if (pos[0] > 94)
       *pn2 = (pos[0] - 94) / 40 + 3;
     else
      {
       *pn1 = 0;
       prclcurs (0);
       if (*pn4 == 1)
        {
         *pn4 = 0;
         writegr (77,28,"DEUX\0");
        }
       else
        {
         *pn4 = 1;
         writegr (77,28,"TOUS\0");
        }
       prclcurs (1);
      }
    }
   if (*pn1 == 4)
     mendel (pn2,pn3);
  }
 prclcurs (0);
 setgrs ();
 selcurs (1);
 restmolec ();
 prclcurs (1);
}



/*************************************************************************/
/* Ecriture d'une chaine de caractères à la position graphique X,Y sur   */
/* l'écran. Les caractères sont écrits dans une matrice de 3x5 points.   */
/* La chaîne doit se terminer par '\0'.                                  */
/*************************************************************************/

writegr (x,y,pc)
int x,y;
char *pc;
 {
  while (*pc != '\0')
   {
    prcargr (x,y,*pc);
    pc++;
```

```
    x += 4;
   }
 }



/*************************************************************************/
/* Choix d'un atome parmi le tableau de Mendeleev. L'atome choisi est    */
/* codé sur deux caractères, le deuxième pouvant éventuellement contenir*/
/* la valeur 0 si le symbole ne comporte qu'une lettre.                  */
/* Attention, cette procédure est directement liée à la définition de    */
/* l'écran.                                                              */
/*************************************************************************/

mendel (pn1,pn2)
char *pn1,*pn2;
 {
  int i,j;
  int x,y;
  int pos [2];
  char *p[9];
  char *d[9];
  char *ptr;

  p[1] = "H                  H";
  p[2] = "LB            BCNOFN";
  p[3] = "NM            ASPSCA";
  p[4] = "KCSTVCMFCNCZGGASBK";
  p[5] = "RSYZNMTRRPACISSTIX";
  p[6] = "CBLHTWROIPAHTPBPAR";
  p[7] = "FRA CPNPSEGTDHETYL";
  p[8] = "    TPUNPACBCEFMNL";
  d[1] = "                   e";
  d[2] = "ie                 e";
  d[3] = "ag           li   lr";
  d[4] = " aci rneoiunaeserr";
  d[5] = "br rbocuhdgdnnbe e";
  d[6] = "saafa esrtuglbiotn";
  d[7] = "rac erdmmudbyormbu";
  d[8] = "    ha pummkfsmdow";
  prclcurs (0);
  setgrs ();
  hline (6,23,3);
  hline (295,312,3);
  hline (6,40,27);
  hline (210,312,27);
  hline (6,40,51);
  hline (210,312,51);
  for (y = 75; y < 148; y += 24)
    hline (6,312,y);
  hline (6,57,171);
  hline (74,312,148);
  hline (74,312,171);
  hline (74,312,195);

  vline (6,3,171);
  vline (23,3,171);
  vline (40,27,171);
  vline (57,75,171);
  for (x = 74; x < 194; x += 17)
    vline (x,75,195);
  vline (75,147,195);
  for (x = 210; x < 279; x += 17)
    vline (x,27,195);
```

```
 vline (295,3,195);
 vline (312,3,195);

 y = 15;
 for (i = 1; i < 9; i++)
  {
   x = 12;
   ptr = p[i];
   for (j = 1; j < 19; j++)
    {
     prcargr (x,y,*ptr);
     x = x + 17;
     ptr++;
    }
   y = y + 24;
  }
 y = 15;
 for (i = 1; i < 9; i++)
  {
   x = 16;
   ptr = d[i];
   for (j = 1; j < 19; j++)
    {
     prcargr (x,y,*ptr);
     x = x + 17;
     ptr++;
    }
   y = y + 24;
  }
 windcurs (7,311,4,194);
 prclcurs (1);
 *pn1 = ' ';
 while (*pn1 == ' ')
  {
   while (getposc (pos) != 0);
   while (getposc (pos) == 0);
   x = (pos[0] - 6) / 17;
   y = (pos[1] - 3) / 24 + 1;
   ptr = p[y];
   while (x-- != 0)
     ptr++;
   *pn1 = *ptr;
   *pn2 = *(ptr - p[y] + d[y]);
  }
 if (*pn2 == ' ')
   *pn2 = 0;
}



/*************************************************************************/
/* Positionnement d'un cycle sur le dessin. Ce cycle fera partie du      */
/* dessin si le bouton gauche de la souris est pressé. Si le cycle est   */
/* superposé à un autre cycle de même taille, le cycle existant sera     */
/* effacé si le bouton droit de la souris est enfoncé.                   */
/* La taille du cycle est donnée par la valeur de n qui peut varier de   */
/* 3 à 8 pour une taille de cycle correspondante.                        */
/* Si deform = 1, le cycle pourra être déformé.                          */
/*************************************************************************/

plcycle (n,deform)
int n,deform;
 {
  int angle1;
```

```
int lx [tmcycle],lx1 [tmcycle];
int ly [tmcycle],ly1 [tmcycle];
int lcouple [dtmcycle];
int x,y;
int pos [2],pos1 [2];
int e;
int i,j,n1;
int adj,adjc,ok;
int inter;

anglel = 360/n;
lx [1] = lx [0] = ly [0] = 0;
ly [1] = ll;
n1 = n - 1;
for (i = 2; i < n; i++)
 {
  lx [i] = vlsin[(i - 1) * anglel] + lx[i - 1];
  ly [i] = vlcos[(i - 1) * anglel] + ly[i - 1];
 }
prclcurs (0);
selcurs (2);
getposc (pos);
x = pos [0];
y = pos [1];
for (i = 0; i < n1; i++)
  line (lx [i]+x, ly [i]+y, lx [i+1]+x, ly [i+1]+y, 1, 0);
line (lx [n1]+x, ly [n1]+y, x, y, 1, 0);
prclcurs (1);

while (((inter = getposc (pos)) != 0) && (inter != 3))
  if ((pos [0] != x) || (pos [1] != y))
    transcycle (n,pos,lx,ly,&x,&y);

adj = 0;
if ((adj = proch1som (x,y,pos1,&e)) != 0)
  transcycle (n,pos1,lx,ly,&x,&y);
for (i = 0; i < n; i++)
 {
  lx [i] += x;
  ly [i] += y;
 }
pos [0] = x;
pos [1] = y + ll;
prclcurs (0);
windcurs (x - ll, x + ll, y - ll, y + ll);
y = y + ll;
putposc (pos);
prclcurs (1);

while (getposc (pos) == 0)
  if ((pos [0] != x) || (pos [1] != y))
    rotatcycle (n,anglel,pos,lx,ly,&x,&y,0);

if ((adj == 1) && ((adj = proch2som (lx[0],ly[0],x,y,pos1,&e)) == 1))
  rotatcycle (n,anglel,pos1,lx,ly,&x,&y,1);

if ((adjc = adjcycle (n,lx,ly,lx1,ly1)) == 1)
 {
  prclcurs (0);
  tracycle (n,lx,ly,0,0);
  tracycle (n,lx1,ly1,1,0);
  prclcurs (1);
  for (i = 2; i < n; i++)
   {
```

```
    lx[i] = lx1[i];
    ly[i] = ly1[i];
   }
 }

for (i = 0; i < 1000; i++);
inter = getposc (pos);

if (inter == 1)
 {
  if ((adjc == 0) && (adj == 1) && (deform == 1))
    if (defocycle (n,lx,ly,lx1,ly1) == 1)
     {
      prclcurs (0);
      tracycle (n,lx,ly,0,0);
      tracycle (n,lx1,ly1,1,0);
      prclcurs (1);
      for (i = 2; i < n; i++)
       {
        lx[i] = lx1[i];
        ly[i] = ly1[i];
       }
     }
  ok = 1;
  i = 1;
  while ((ok == 1) && (i < n))
    if (prochsoml (lx[i],ly[i++],pos1,&e) != 0)
      if (existsom (pos1[0],pos1[1]) != 1)
        ok = 0;
  i = 1;
  if (adj != 1)
    while ((ok == 1) && (i < n))
      if (prochlsom (lx[i],ly[i++],pos1,&e) != 0)
        ok = 0;
 }

if (inter == 2)
  if ((adjc == 0) && (adj == 1))
    if (defocycle (n,lx,ly,lx1,ly1) != 0)
     {
      prclcurs (0);
      tracycle (n,lx,ly,0,0);
      tracycle (n,lx1,ly1,1,0);
      prclcurs (1);
      for (i = 2; i < n; i++)
       {
        lx[i] = lx1[i];
        ly[i] = ly1[i];
       }
     }

j = 0;
for (i = 0; i < n; i++)
 {
  lcouple [j++] = lx [i];
  lcouple [j++] = ly [i];
 }

if ((inter == 1) && (ok == 1))
 {
  if (addcycle (n,lcouple) == 0)
   {
    prclcurs (0);
    tracycle (n,lx,ly,2,0);
```

```
    restatlp (n,lcouple);
    prclcurs (1);
   }
  else
   {
    prclcurs (0);
    tracycle (n,lx,ly,0,0);
    prclcurs (1);
   }
 }
else if (inter == 2)
 {
  if (supprcycle (n,lcouple) == 0)
   {
    prclcurs (0);
    tracycle (n,lx,ly,2,0);
    tracycle (n,lx,ly,0,0);
    for (i = 0; i < n1; i++)
     {
      if (trouvlicy (lx[i],ly[i],lx[i+1],ly[i+1]) == 1)
       {
        line (lx[i],ly[i],lx[i+1],ly[i+1],1,1);
        line (lx[i],ly[i],lx[i+1],ly[i+1],2,1);
       }
      else
       {
        efflm (lx[i],ly[i],lx[i+1],ly[i+1]);
        if (existsom (lx[i],ly[i]) != 0)
          hline (lx[i],lx[i],ly[i]);
        if (existsom (lx[i+1],ly[i+1]) != 0)
          hline (lx[i+1],lx[i+1],ly[i+1]);
       }
     }
    if (trouvlicy (lx[n1],ly[n1],lx[0],ly[0]) == 1)
     {
      line (lx[n1],ly[n1],lx[0],ly[0],1,1);
      line (lx[n1],ly[n1],lx[0],ly[0],2,1);
     }
    else
      efflm (lx[n1],ly[n1],lx[0],ly[0]);
    restatls (n,lcouple);
    prclcurs (1);
   }
  else
   {
    prclcurs (0);
    tracycle (n,lx,ly,0,0);
    prclcurs (1);
   }
 }
else
 {
  prclcurs (0);
  tracycle (n,lx,ly,0,0);
  prclcurs (1);
 }
prclcurs (0);
selcurs (1);
prclcurs (1);
}
```

```
/*************************************************************************/
/* Effectue la translation du cycle de taille n dont les coordonnées     */
/* relatives des sommets sont dans lx et ly. Les coordonnées réelles     */
/* sont obtenues en ajoutant *px et *py à ces coordonnées.               */
/* Le tableau pos doit contenir la position du curseur où devra être     */
/* placé le sommet (lx[0],ly[0]).                                        */
/*************************************************************************/

transcycle (n,pos,lx,ly,px,py)
int n;
int pos[], lx[], ly[];
int *px, *py;
 {
  int n1,i;

  n1 = n - 1;
  prclcurs (0);
  for (i = 0; i < n1; i++)
    line (lx [i]+*px, ly [i]+*py, lx [i+1]+*px, ly [i+1]+*py, 0, 0);
  line (lx [n1]+*px, ly [n1]+*py, *px, *py, 0, 0);
  *px = pos [0];
  *py = pos [1];
  for (i = 0; i < n1; i++)
    line (lx [i]+*px, ly [i]+*py, lx [i+1]+*px, ly [i+1]+*py, 1, 0);
  line (lx [n1]+*px, ly [n1]+*py, *px, *py, 1, 0);
  prclcurs (1);
 }



/*************************************************************************/
/* Effectue la rotation d'un cycle de taille n. Les tableaux lx et ly    */
/* contiennent les coordonnées des sommets (lx[i],ly[i]) avant rotation. */
/* Le tableau pos contient la position courante du curseur. Ses          */
/* coordonnées doivent être dans un carré centré sur (lx[0],ly[0]) et    */
/* de côté 2 * ll + 1.                                                   */
/* En sortie, les tableaux lx et ly contiennent les nouvelles coordonnées */
/* des sommets du cycles et le cycle est adapté à la rotation demandée.  */
/* De plus, la position du curseur est modifiée pour qu'il pointe        */
/* toujours le même sommet du cycle.                                     */
/*************************************************************************/

rotatcycle (n,anglel,pos,lx,ly,px,py,fixe)
int n,anglel;
int pos[],lx[],ly[];
int *px,*py;
int fixe;
 {
  int n1,i;
  int angle,anglec;
  int xt,yt;

  n1 = n - 1;
  anglec = trouvalpha (lx [0],ly [0],pos [0],pos [1]);
  if (fixe == 0)
   {
    xt = vlsin [anglec] + lx[0];
    yt = vlcos [anglec] + ly[0];
   }
  else
   {
    xt = pos[0];
    yt = pos[1];
```

```
   }
  prclcurs (0);
  tracycle (n,lx,ly,0,0);
  lx[1] = xt;
  ly[1] = yt;
  for (i = 2; i < n; i++)
   {
    angle = (i - 1) * anglel + anglec;
    while (angle > 359)
      angle = angle - 360;
    lx [i] = vlsin [angle] + lx [i-1];
    ly [i] = vlcos [angle] + ly [i-1];
   }
  tracycle (n,lx,ly,1,0);
  pos [0] = *px = lx [1];
  pos [1] = *py = ly [1];
  putposc (pos);
  prclcurs (1);
 }



/*************************************************************************/
/* Trace un cycle de taille n dont les coordonnées réelles sont dans     */
/* lx et ly. Les paramètres p1 et p2 sont ceux acceptés par LINE.        */
/*************************************************************************/

tracycle (n,lx,ly,p1,p2)
int n,p1,p2;
int lx[],ly[];
 {
  int i;

  n--;
  for (i = 0; i < n; i++)
    line (lx [i], ly [i], lx [i+1], ly [i+1], p1, p2);
  line (lx [n], ly [n], lx [0], ly [0], p1, p2);
 }



/*************************************************************************/
/* Ajoute les coordonnées des sommets d'un cycle de taille n dans la table */
/* cycle s'il n'existait pas déjà. Les coordonnées des sommets doivent   */
/* être dans le tableau liste.                                           */
/* La fonction retourne la valeur 0 si tout s'est bien passé, 1 si la place*/
/* restante est insuffisante pour accueillir le nouveau cycle et 2 si le */
/* cycle existait déjà.                                                  */
/* contenu du tableau CYCLE pour chaque cycle :                          */
/*          taille du cycle                                              */
/*          coordonnée X premier sommet                                  */
/*          coordonnée Y premier sommet                                  */
/*          ...                                                          */
/* Les sommets sont décrit dans le sens antihorlogique.                  */
/*************************************************************************/

addcycle (n,liste)
int n;
int liste [];
 {
  int i,n1;
  int ok,p;

  ok = 2;
```

```
 if (trouvcycle (n,liste,&p) == 0)
  {
   n1 = 2 * n;
   if ((cyclsuiv + n1 + 2) > maxsom)
     ok = 1;
   else
    {
     ok = 0;
     cycle [cyclsuiv++] = n;
     i = 0;
     while (i < n1)
       cycle [cyclsuiv++] = liste [i++];
    }
  }
 return (ok);
}



/*****************************************************************************/
/* Supprime un cycle de taille n de la table cycle. Si tout c'est bien       */
/* passé, la fonction retourne la valeur 0 sinon 1 si le cycle n'existait    */
/* pas.                                                                      */
/*****************************************************************************/

supprcycle (n,liste)
int n;
int liste [];
 {
  int i,j;
  int p,ok;

  ok = 1;
  if (trouvcycle (n,liste,&p) == 1)
   {
    ok = 0;
    i = p;
    j = p + 2 * cycle[p] + 1;
    while (j < cyclsuiv)
      cycle [i++] = cycle [j++];
    cyclsuiv = i;
   }
  return (ok);
 }



/*****************************************************************************/
/* Recherche d'un cycle dont les coordonnées sont dans liste. La valeur      */
/* de la fonction est 1 si le cycle à été trouvé sinon 0.                    */
/* pp pointe le cycle trouvé.                                                */
/*****************************************************************************/

trouvcycle (n,liste,pp)
int n;
int liste [];
int *pp;
 {
  int i,j,sj;
  int n1,n2;
  int ok,trouve;

  i = 0;
  trouve = 0;
```

```
  n2 = 2 * n;
  while ((i < cyclsuiv) && (trouve == 0))
   {
    n1 = cycle [i++];
    if (n1 != n)
      i += 2 * n1;
    else
     {
      *pp = i - 1;
      j = 0;
      ok = 1;
      while ((j < n2) && (ok == 1) && (trouve == 0))
        if ((cycle[i] == liste[j++]) && (cycle [i+1] == liste [j++]))
         {
          sj = j - 2;
          i += 2;
          while ((j < n2) && (ok == 1))
            if ((cycle[i++] != liste[j++]) || (cycle[i++] != liste[j++]))
              ok = 0;
          j = 0;
          while ((j < sj) && (ok == 1))
            if ((cycle[i++] != liste[j++]) || (cycle[i++] != liste[j++]))
              ok = 0;
          if (ok == 1)
            trouve = 1;
         }
      i = *pp + n2 + 1;
     }
   }
  return (trouve);
 }



/*************************************************************************/
/* Recherche d'un sommet proche du sommet de coordonnées (x,y). Les tables */
/* cycle et sommet sont consultées et le sommet le plus proche est retenu  */
/* s'il existe. Dans ce cas, ces coordonnées sont passées dans couple et   */
/* la valeur de la fonction est 1 si sommet d'un cycle ou 2 si sommet      */
/* d'une liaison sinon, elle est 0. Le pointeur pe pointe vers la variable */
/* contenant l'écart entre les deux points si trouvé.                      */
/*************************************************************************/

prochisom (x,y,couple,pe)
int x,y;
int couple [];
int *pe;
 {
  int i,n;
  int e,e1,e2;
  long int el,el1,el2;
  int couple2 [2];
  int min,pos;
  int trouve;

  trouve = 0;
  *pe = min = 1000;
  i = 0;
  while (i < cyclsuiv)
   {
    n = cycle [i++];
    n = 2 * n + i;
    while (i < n)
     {
```

```
      e1 = x - cycle [i++];
      e2 = y - cycle [i++];
      e1 = (e1 < 0) ? -e1 : e1;
      e2 = (e2 < 0) ? -e2 : e2;
      *pe = e1 + e2;
      if (*pe < min)
       {
        min = *pe;
        pos = i - 1;
       }
     }
   }
  if (min < 1000)
   {
    couple [1] = cycle [pos];
    couple [0] = cycle [--pos];
    el1 = x - couple[0];
    el2 = y - couple[1];
    el = el1 * el1 + el2 * el2;
    if (el < prochval)
     {
      trouve = 1;
      *pe = el;
     }
    else
      *pe = 1000;
   }
  if (prochsom1 (x,y,couple2,&e) == 1)
    if (e < *pe)
     {
      *pe = e;
      couple[0] = couple2[0];
      couple[1] = couple2[1];
      trouve = 2;
     }
  return (trouve);
 }



/*************************************************************************/
/* Recherche d'un sommet proche de x1 et y1. Seule la table cycle est    */
/* consultée. De plus, le sommet trouvé doit appartenir à un cycle       */
/* contenant le sommet x0,y0. Si un sommet est trouvé, la valeur         */
/* retournée par la fonction est 1 et les coordonnées du sommet seront   */
/* dans pos. Sinon, la valeur de la fonction est 0.                      */
/*************************************************************************/

proch2som (x0,y0,x1,y1,couple,pe)
int x0,y0,x1,y1;
int couple [];
int *pe;
 {
  int i,si,n,p,pos,pi;
  int e,e1,e2,min;
  long int el,el1,el2;
  int trouve,fin;

  trouve = 0;
  min = 1000;
  i = 0;
  while (i < cyclsuiv)
   {
    n = cycle [i++];
```

```
  n = 2 * n + i;
  si = i;
  fin = 0;
  while (fin == 0)
   {
    while ((i < n) && (cycle[i++] != x0))
      i++;
    if ((i < n) && (cycle[i++] == y0))
     {
      fin = 1;
      p = i - 2;
      if (i == n)
        i = si;
      e1 = x1 - cycle [i];
      e2 = y1 - cycle [++i];
      e1 = (e1 < 0) ? -e1 : e1;
      e2 = (e2 < 0) ? -e2 : e2;
      *pe = e1 + e2;
      pi = i;
      i = p;
      if (i == si)
        i = n;
      i = i - 2;
      e1 = x1 - cycle [i];
      e2 = y1 - cycle [++i];
      e1 = (e1 < 0) ? -e1 : e1;
      e2 = (e2 < 0) ? -e2 : e2;
      if ((e1 + e2) < *pe)
       {
        *pe = e1 + e2;
        pi = i;
       }
      if (*pe < min)
       {
        min = *pe;
        pos = pi;
       }
     }
    else if (i == n)
      fin = 1;
   }
  i = n;
 }
 if (min < 1000)
  {
   couple [1] = cycle [pos];
   couple [0] = cycle [--pos];
   el1 = x1 - couple[0];
   el2 = y1 - couple[1];
   el = el1 * el1 + el2 * el2;
   if (el < prochval)
    {
     trouve = 1;
     *pe = el;
    }
  }
 return (trouve);
}
```

```
/*************************************************************************/
/* Cette procédure regarde s'il n'existe pas un cycle qui se supperpose à  */
/* peu près au cycle dont les coordonnées des sommets sont dans lx et ly.  */
/* Si c'est le cas, les coordonnées de ce cycle sont retournées par les    */
/* tableaux lxf et lyf et la valeur de la fonction est 1 sinon 0.          */
/*************************************************************************/

adjcycle (n,lx,ly,lxf,lyf)
int n;
int lx[],ly[];
int lxf[],lyf[];

 {
  int i,j,d1,d2,si;
  int n1,n2;
  int trouve,fin;

  i = 0;
  trouve = 0;
  lxf[0] = lx[0];
  lyf[0] = ly[0];
  lxf[1] = lx[1];
  lyf[1] = ly[1];
  while ((i < cyclsuiv) && (trouve == 0))
   {
    n1 = cycle [i++];
    n2 = i + 2 * n1;
    if (n1 != n)
      i = n2;
    else
     {
      si = i;
      fin = 0;
      while (fin == 0)
       {
        while ((i < n2) && (lx[0] != cycle[i++]))
          i++;
        if ((i < n2) && (ly[0] == cycle[i++]))
         {
          fin = 1;
          if (i == n2)
            i = si;
          if ((lx[1] == cycle[i++]) && (ly[1] == cycle[i++]))
           {
            trouve = 1;
            j = 2;
            if (i == n2)
              i = si;
            while ((j < n) && (trouve == 1))
             {
              d1 = cycle[i++] - lx[j];
              d2 = cycle[i] - ly[j];
              if ((d1*d1 + d2*d2) < adjval)
               {
                lxf[j] = cycle[i-1];
                lyf[j] = cycle[i];
               }
              else
                trouve = 0;
              j++;
              i++;
              if (i == n2)
```

```
                i = si;
              }
            }
          }
        else if (i == n2)
          fin = 1;
       }
      i = n2;
     }
   }
  return (trouve);
 }


/*************************************************************************/
/* Cette procédure recherche des sommets de cycles suffisamment proches de*/
/* ceux d'un cycle donné et les y identifie. Cela permet de rendre commun */
/* à d'autres cycles plus de deux sommets. Attention, cette procédure ne  */
/* peut être appelée que si le cycle donné à déjà deux sommets en commun  */
/* avec un même cycle. On risquerait autrement d'être en opposition avec  */
/* certaines règles.                                                      */
/* La fonction vaut 0 si aucun sommet n'a été modifié, 2 si au moins deux */
/* des sommets modifiés l'on été par rapport à un cycle identique et 1    */
/* dans les autres cas. Si la fonction vaut 2, on a en fait un pontage de */
/* cycle.                                                                 */
/*************************************************************************/

defocycle (n,lx,ly,lxf,lyf)
int n;
int lx[],ly[];
int lxf[],lyf[];
 {
  int i,j,ccour;
  int n1;
  int d1,d2,d;
  int dmin,jmin,cycmin;
  int idcycle [tmcycle];
  int trouve;

  i = 0;
  while (i < n)
   {
    lxf[i] = lx[i];
    lyf[i] = ly[i++];
   }
  i = 2;
  trouve = 0;
  while (i < n)
   {
    j = 0;
    dmin = 1000;
    while ((j < cyclsuiv) && (dmin != 0))
     {
      ccour = j;
      n1 = 2 * cycle[j++] + j;
      while (j < n1)
       {
        d1 = cycle[j++] - lx[i];
        d2 = cycle[j++] - ly[i];
        d1 = (d1 < 0) ? -d1 : d1;
        d2 = (d2 < 0) ? -d2 : d2;
        d = d1 + d2;
        if (d < dmin)
```

```
        {
         dmin = d;
         jmin = j;
         cycmin = ccour;
        }
       }
     }
    d1 = cycle[jmin-2] - lx[i];
    d2 = cycle[jmin-1] - ly[i];
    if ((d1*d1 + d2*d2) < maxdef)
     {
      lxf[i] = cycle[jmin-2];
      lyf[i] = cycle[jmin-1];
      idcycle[i] = cycmin;
      trouve = 1;
     }
    else
     idcycle[i] = cyclsuiv;
    i++;
   }
  n = n - 1;
  i = 2;
  while ((i < n) && (trouve == 1))
   {
    ccour = idcycle[i];
    if (ccour != cyclsuiv)
      if (somcycle(lx[0],ly[0],ccour,lxf[i],lyf[i]) == 1)
        if (somcycle(lx[1],ly[1],ccour,lxf[i],lyf[i]) == 1)
          trouve = 2;
    i++;
   }
  return (trouve);
 }



/*************************************************************************/
/* Détermine si le sommet (x,y) appartient au cycle contenant le sommet  */
/* (x1,y1) et se trouvant à la position poscycle dans la table cycle. Si */
/* la recherche échoue, la procédure recherche dans la table le cycle    */
/* suivant qui pourrait à la fois contenir les sommets (x,y) et (x1,y1). */
/*************************************************************************/

somcycle (x,y,poscycle,x1,y1)
int x,y;
int x1,y1;
int poscycle;
 {
  int i,si,n;
  int next,trouve;

  trouve = 0;
  i = poscycle;
  n = cycle[i++] * 2 + i;
  while ((trouve == 0) && (i < cyclsuiv))
   {
    while ((i < n) && (trouve == 0))
     {
      if ((cycle[i++] == x) && (cycle[i] == y))
        trouve = 1;
      else
        i++;
     }
    if (trouve == 0)
```

```
    {
     next = 0;
     i = n;
     while ((next == 0) && (i < cyclsuiv))
      {
       n = cycle[i++] * 2 + i;
       si = i;
       while ((i < n) && (next == 0))
         if ((cycle[i++] == x1) && (cycle[i] == y1))
          {
           next = 1;
           i = si;
          }
         else
           i++;
      }
    }
  }
 return (trouve);
}



/*************************************************************************/
/* On va regarder si la liaison (x0,y0)-(x1,y1) fait partie d'un         */
/* cycle. Si c'est le cas, la valeur retournée par la fonction est 1     */
/* sinon 0.                                                              */
/*************************************************************************/

trouvlicy (x0,y0,x1,y1)
int x0,y0,x1,y1;
 {
  int i,si,n,p;
  int trouve,fin;

  trouve = 0;
  i = 0;
  while ((i < cyclsuiv) && (trouve == 0))
   {
    n = cycle[i++];
    n = 2 * n + i;
    si = i;
    fin = 0;
    while (fin == 0)
     {
      while ((i < n) && (cycle[i++] != x0))
        i++;
      if ((i < n) && (cycle[i++] == y0))
       {
        fin = 1;
        p = i - 2;
        if (i == n)
          i = si;
        if ((cycle[i++] == x1) && (cycle[i++] == y1))
          trouve = 1;
        else
         {
          i = p;
          if (i == si)
            i = n;
          i = i - 2;
          if ((cycle[i++] == x1) && (cycle[i++] == y1))
            trouve = 1;
         }
```

```
      }
     else if (i == n)
       fin = 1;
    }
   i = n;
  }
 return (trouve);
}



/*****************************************************************************/
/* Placement d'une nouvelle liaison simple si celle-ci n'existait pas.       */
/* Il est également possible d'effacer une liaison existante.                */
/*****************************************************************************/

plsliais ()
 {
  int x,y,x0,y0,x1,y1;
  int inter;
  int angle;
  int pos [2];
  int pos1 [2];
  int lcouple [4];
  int i,e,e1,e2;

  prclcurs (0);
  selcurs (2);
  getposc (pos);
  x = pos[0];
  y = pos [1];
  line (x,y,x+11,y,1);
  prclcurs (1);

  while (((inter = getposc(pos)) != 0) && (inter != 3))
    if ((pos[0] != x) || (pos[1] != y))
     {
      prclcurs (0);
      line (x,y,x+11,y,0,1);
      x = pos [0];
      y = pos [1];
      line (x,y,x+11,y,1,1);
      prclcurs (1);
     }

  if (proch1som (x,y,pos1,&e) != 0)
   {
    x0 = pos1[0];
    y0 = pos1[1];
   }
  else
   {
    x0 = x;
    y0 = y;
   }

  x1 = x0 + 11;
  y1 = y0;
  if (proch1som (x0+11,y0,pos1,&e1) != 0)
   {
    if (proch1som (x0-11,y0,pos1,&e2) != 0)
     {
      if (e1 < e2)
        x1 = x0 - 11;
```

```
    }
   else
     x1 = x0 - l1;
 }

prclcurs (0);
line (x,y,x+l1,y,0,1);
line (x0,y0,x1,y1,1,1);
pos [0] = x1;
pos [1] = y1;
putposc (pos);
windcurs (x0-l1, x0+l1, y0-l1, y0+l1);
prclcurs (1);

while (getposc(pos) == 0)
  if ((pos[0] != x1) || (pos[1] != y1))
    {
     angle = trouvalpha (x0,y0,pos[0],pos[1]);
     x = vlsin [angle] + x0;
     y = vlcos [angle] + y0;
     prclcurs (0);
     line (x0,y0,x1,y1,0,1);
     x1 = x;
     y1 = y;
     line (x0,y0,x1,y1,1,1);
     pos [0] = x1;
     pos [1] = y1;
     putposc (pos);
     prclcurs (1);
    }
lcouple[0] = x0;
lcouple[1] = y0;
lcouple[2] = x1;
lcouple[3] = y1;

for (i = 0; i < 1000; i++);
inter = getposc (pos);

if ((inter == 1) && (prochlsom (x1,y1,pos1,&e) == 0))
 {
  if ((trouvlicy (x0,y0,x1,y1) == 0) && (addliais (x0,y0,x1,y1) == 0))
    {
     prclcurs (0);
     line (x0,y0,x1,y1,2,1);
     restatlp (2,lcouple);
     prclcurs (1);
    }
   else
    {
     prclcurs (0);
     line (x0,y0,x1,y1,0,1);
     prclcurs (1);
    }
 }
else if (inter == 2)
 {
  if (prochsoml (x1,y1,pos1,&e) == 1)
    {
     prclcurs (0);
     line (x0,y0,x1,y1,0,1);
     lcouple[2] = x1 = pos1 [0];
     lcouple[3] = y1 = pos1 [1];
     line (x0,y0,x1,y1,1,1);
     prclcurs (1);
```

```
    }
   if (supprliais (x0,y0,x1,y1) == 0)
    {
     prclcurs (0);
     line (x0,y0,x1,y1,2,1);
     line (x0,y0,x1,y1,0,1);
     efflm (x0,y0,x1,y1);
     if (existsom (x0,y0) != 0)
       hline (x0,x0,y0);
     if (existsom (x1,y1) != 0)
       hline (x1,x1,y1);
     restatls (2,lcouple);
     prclcurs (1);
    }
   else
    {
     prclcurs (0);
     line (x0,y0,x1,y1,0,1);
     prclcurs (1);
    }
  }
 else
  {
   prclcurs (0);
   line (x0,y0,x1,y1,0,1);
   prclcurs (1);
  }
 prclcurs (0);
 selcurs (1);
 prclcurs (1);
}



/*************************************************************************/
/* Ajout d'une liaison dans la table sommet si elle n'était pas déjà     */
/* existante. Les coordonnées des deux sommets sont x0,y0 et x1 et y1.   */
/* Si tout s'est bien passé, la valeur retournée par la fonction est 0.  */
/* Elle est de 1 s'il ne reste pas assez de place pour stocker la        */
/* liaison et enfin de 2 si la liaison existait déjà.                    */
/* Contenu du tableau SOMMET :                                           */
/*          coordonnées X,Y du premier sommet de la liaison              */
/*          coordonnées X,Y du deuxième sommet                           */
/* somsuiv pointe la première place libre dans sommet.                   */
/*************************************************************************/

addliais (x0,y0,x1,y1)
int x0,y0,x1,y1;
 {
  int p;
  int ok;

  ok = 2;
  if (trouvliais (x0,y0,x1,y1,&p) == 0)
   {
    ok = 1;
    if ((somsuiv + 4) < maxsom)
     {
      ok = 0;
      sommet [somsuiv++] = x0;
      sommet [somsuiv++] = y0;
      sommet [somsuiv++] = x1;
      sommet [somsuiv++] = y1;
     }
```

```
   }
  return (ok);
 }



/*************************************************************************/
/* Supprime une liaison dans la table sommet.                            */
/* Si tout s'est bien passé, la valeur retournée par la fonction est 0   */
/* sinon 1 si la liaison n'existait pas.                                 */
/*************************************************************************/

supprliais (x0,y0,x1,y1)
int x0,y0,x1,y1;
 {
  int i,p;
  int ok;

  ok = 1;
  if (trouvliais (x0,y0,x1,y1,&p) == 1)
   {
    ok = 0;
    i = p + 4;
    while (i < somsuiv)
      sommet [p++] = sommet [i++];
    somsuiv = p;
   }
  return (ok);
 }



/*************************************************************************/
/* Recherche d'une liaison dans la table sommet. La valeur retournée par */
/* la fonction est 1 si la liaison a été trouvée sinon 0. Si la liaison  */
/* est trouvée, la position du premier sommet dans sommet est pp.        */
/*************************************************************************/

trouvliais (x0,y0,x1,y1,pp)
int x0,y0,x1,y1;
int *pp;
 {
  int i;
  int x,y;
  int ok,trouve;

  i = 0;
  trouve = 0;
  while ((i < somsuiv) && (trouve == 0))
   {
    ok = 0;
    if ((sommet[i] == x0) && (sommet[i+1] == y0))
     {
      x = x1;
      y = y1;
      ok = 1;
     }
    else if ((sommet[i] == x1) && (sommet[i+1] == y1))
     {
      x = x0;
      y = y0;
      ok = 1;
     }
    if ((ok == 1) && (sommet[i+2] == x) && (sommet[i+3] == y))
```

```
    {
     trouve = 1;
     *pp = i;
    }
   i = i + 4;
  }
 return (trouve);
}



/*************************************************************************/
/* Recherche d'un sommet 'proche' du sommet (x,y) dans la table sommet.  */
/* Si un sommet est trouvé, ses coordonnées sont placées dans le tableau */
/* couple et la fonction retourne le valeur 1 et pe est l'écart entre les */
/* deux sommets sinon la valeur de fonction retournée est 0.             */
/*************************************************************************/

prochsoml (x,y,couple,pe)
int x,y;
int couple [];
int *pe;
 {
  int trouve;
  int min,e1,e2,pos;
  long int el,el1,el2;
  int i;

  trouve = 0;
  min = 1000;
  for (i = 0; i < somsuiv; ++i)
   {
    e1 = x - sommet [i];
    e2 = y - sommet [++i];
    e1 = (e1 < 0) ? -e1 : e1;
    e2 = (e2 < 0) ? -e2 : e2;
    *pe = e1 + e2;
    if (*pe < min)
     {
      min = *pe;
      pos = i;
     }
   }
  if (min < 1000)
   {
    couple [1] = sommet [pos];
    couple [0] = sommet [--pos];
    el1 = x - couple[0];
    el2 = y - couple[1];
    el = el1 * el1 + el2 * el2;
    if (el < prochval)
     {
      trouve = 1;
      *pe = el;
     }
   }
 return (trouve);
}
```

```
/*****************************************************************************/
/* Placement d'une liaison multiple. On part d'une liaison simple, on passe */
/* à une double puis une triple. Si on continue, on revient à une liaison   */
/* simple.                                                                   */
/*****************************************************************************/

plmliais ()
 {
  int pos [2], pos1 [2];
  int lcouple [4];
  int inter, e, r;
  int x,y,x1,y1;
  int i;

  getposc (pos);
  if (prochisom (pos[0],pos[1],pos1,&e) != 0)
   {
    prclcurs (0);
    selcurs (2);
    x = pos1[0];
    y = pos1[1];
    x1 = pos[0];
    y1 = pos[1];
    windcurs (x-11,x+11,y-11,y+11);
    line (x,y,x1,y1,1,1);
    prclcurs (1);
    while (((inter = getposc (pos)) != 0) && (inter != 3))
      if ((pos[0] != x1) || (pos[1] != y1))
       {
        prclcurs (0);
        line (x,y,x1,y1,0,1);
        x1 = pos[0];
        y1 = pos[1];
        line (x,y,x1,y1,1,1);
        prclcurs (1);
       }
    prclcurs (0);
    line (x,y,x1,y1,0,1);
    selcurs (1);
    prclcurs (1);
    if (prochisom (x1,y1,pos1,&e) != 0)
     {
      lcouple[0] = x;
      lcouple[1] = y;
      lcouple[2] = x1 = pos1[0];
      lcouple[3] = y1 = pos1[1];
      if (inter != 3)
        if ((trouvlicy (x,y,x1,y1) == 1) || (trouvliais (x,y,x1,y1) == 1))
         {
          r = inclm (x,y,x1,y1);
          if (r == 2)
            trml (x,y,x1,y1,1,2);
          else if (r == 3)
            trml (x,y,x1,y1,1,3);
          else if (r == 0)
           {
            trml (x,y,x1,y1,0,2);
            trml (x,y,x1,y1,0,3);
           }
          restatlp (4,lcouple);
         }
     }
   }
```

```
  else
    for (i = 0; i < 10000; i++);
 }



/*****************************************************************************/
/* On trace une liaison multiple si sr = 1. La liaison deviendra double si   */
/* m = 2 et elle deviendra triple si m = 3. (pour passer à une liaison       */
/* double, elle doit initialement être simple et pour passer à une triple,   */
/* elle doit initialement être double)                                       */
/* Si sr = 0, la liaison multiple correspondante est effacée.                */
/*****************************************************************************/

trml (x0,y0,x1,y1,sr,m)
int x0,y0,x1,y1;
int sr,m;
 {
  int xld0,yld0,xld1,yld1;
  int angle;
  int x,y;
  int dx,dy;

  if ((x0 > x1) || ((x0 == x1) && (y0 > y1)))
   {
    x = x0;
    x0 = x1;
    x1 = x;
    y = y0;
    y0 = y1;
    y1 = y;
   }
  angle = trouvalpha (x0,y0,x1,y1) + anglml;
  while (angle > 359)
    angle -= 360;
  dx = vlsin[angle] * dlm / ll;
  dy = vlcos[angle] * dlm / ll;
  if (m == 2)
   {
    xld0 = x0 + dx;
    yld0 = y0 + dy;
    xld1 = x1 + dy;
    yld1 = y1 - dx;
   }
  else
   {
    xld0 = x0 - dy;
    yld0 = y0 + dx;
    xld1 = x1 - dx;
    yld1 = y1 - dy;
   }
  prclcurs (0);
  line (xld0,yld0,xld1,yld1,sr,1);
  if (sr == 1)
    line (xld0,yld0,xld1,yld1,2,1);
  prclcurs (1);
 }
```

```
/*************************************************************************/
/* Ajoute une liaison multiple dans la table lmsommet si elle n'existait */
/* pas. Et ce sera une liaison double. Si la liaison existait déjà, elle */
/* devient triple si elle était double et elle est supprimée si elle était */
/* triple.                                                               */
/* Valeur de la fonction : 2 si formation d'une liaison double           */
/*                         3 si formation d'une liaison triple           */
/*                         0 si la liaison est supprimée                 */
/*                         4 s'il n'y a plus de place dans la table      */
/*                         5 si la liaison simple n'existe pas           */
/* Contenu de LMSOMMET :                                                 */
/*       multiplicité de la liaison                                      */
/*       coordonnées des deux sommets impliqués                          */
/*************************************************************************/

inclm (x0,y0,x1,y1)
int x0,y0,x1,y1;
 {
  int i,p;
  int x,y;
  int existe,ok;

  if ((x0 > x1) || ((x0 == x1) && (y0 > y1)))
   {
    x = x0;
    x0 = x1;
    x1 = x;
    y = y0;
    y0 = y1;
    y1 = y;
   }
  existe = trouvlm (x0,y0,x1,y1,&p);
  if (existe == 2)
   {
    ok = 3;
    lmsommet[p] = 3;
   }
  else if (existe == 3)
   {
    ok = 0;
    supprlm (x0,y0,x1,y1);
   }
  else if (existe == 0)
    if ((trouvliais(x0,y0,x1,y1,&p) == 1) || (trouvlicy (x0,y0,x1,y1) == 1))
     {
      ok = 4;
      i = lmsuiv;
      if ((i + 5) < maxlm)
       {
        ok = 2;
        lmsommet[i++] = 2;
        lmsommet[i++] = x0;
        lmsommet[i++] = y0;
        lmsommet[i++] = x1;
        lmsommet[i++] = y1;
        lmsuiv = i;
       }
     }
  return (ok);
 }
```

```
/*************************************************************************/
/* Suppression d'une liaison multiple dans la table lmsommet. La valeur de  */
/* la fonction est la même que trouvlm.                                      */
/*************************************************************************/

supprlm (x0,y0,x1,y1)
int x0,y0,x1,y1;
 {
  int i,p;
  int existe;

  existe = trouvlm (x0,y0,x1,y1,&p);
  if (existe != 0)
   {
    i = p + 5;
    while (i < lmsuiv)
      lmsommet[p++] = lmsommet[i++];
    lmsuiv = p;
   }
  return (existe);
 }



/*************************************************************************/
/* Effacement d'une liaison multiple dans la table sommet et à l'écran.   */
/*************************************************************************/

efflm (x0,y0,x1,y1)
int x0,y0,x1,y1;
 {
  int m;
  int x,y;

  if ((x0 > x1) || ((x0 == x1) && (y0 > y1)))
   {
    x = x0;
    x0 = x1;
    x1 = x;
    y = y0;
    y0 = y1;
    y1 = y;
   }

  if ((m = supprlm (x0,y0,x1,y1)) != 0)
   {
    trml (x0,y0,x1,y1,0,2);
    if (m == 3)
      trml (x0,y0,x1,y1,0,3);
   }
 }



/*************************************************************************/
/* On recherche s'il existe une liaison multiple entre les sommets        */
/* (x0,y0) - (x1,y1). Si oui, la valeur retoyurnée par la fonction est 2  */
/* ou 3 en fonction du type de liaison et *pp pointe cette liaison. Si    */
/* non, la valeur de la fonction est 0.                                   */
/*************************************************************************/

trouvlm (x0,y0,x1,y1,pp)
```

```
int x0,y0,x1,y1;
int *pp;
 {
  int i,si;
  int x,y;
  int trouve;

  if ((x0 > x1) || ((x0 == x1) && (y0 > y1)))
   {
    x = x0;
    x0 = x1;
    x1 = x;
    y = y0;
    y0 = y1;
    y1 = y;
   }
  trouve = 0;
  i = 0;
  while ((i < lmsuiv) && (trouve == 0))
   {
    si = i;
    if ((lmsommet[++i] == x0) && (lmsommet[++i] == y0))
      if ((lmsommet[++i] == x1) && (lmsommet[++i] == y1))
       {
        trouve = lmsommet [si];
        *pp = si;
       }
    i = si + 5;
   }
  return (trouve);
 }



/*************************************************************************/
/* Placement d'un hétéroatome sur un sommet existant. Si un symbole était  */
/* déjà présent sur ce sommet, il est remplacé par le nouveau. Ces         */
/* opérations sont obtenues en poussant sur le bouton gauche de la souris. */
/* Si on pousse sur le bouton droit, le symbole du sommet choisi est       */
/* simplement supprimé quelque soit l'hétéroatome couramment choisi. Si on */
/* appuie simultanément sur les deux boutons, on ne fait rien.             */
/*************************************************************************/

platome (n1,n2)
char n1,n2;
 {
  int i;
  int e,inter;
  int x,y;
  int pos [2];

  formacur (n1,n2);
  while (getposc (pos) == 0);

  for (i = 0; i < 5000; i++);
  inter = getposc (pos);

  x = pos[0];
  y = pos[1];
  if (inter != 3)
   {
    if (prochisom (x,y,pos,&e) != 0)
     {
      x = pos[0];
```

```
      y = pos[1];
      if (suppratom (x,y) == 0)
        restenvat (x,y);
      if ((inter == 1) && (addatom (x,y,n1,n2) == 0))
        prsymb (x,y,n1,n2);
     }
   }
 }



/*****************************************************************************/
/* Affichage d'un symbole centré sur (x,y). Si le symbole n'a qu'un atome, */
/* n2 doit être égal à 0.                                                    */
/*****************************************************************************/

prsymb (x,y,n1,n2)
int x,y;
int n1,n2;
 {
  prclcurs (0);
  if (n2 == 0)
    prcargr (x,y,n1);
  else
   {
    prcargr (x-2,y,n1);
    prcargr (x+2,y,n2);
   }
  prclcurs (1);
 }



/*****************************************************************************/
/* Efface le symbole centré sur (x,y)                                        */
/*****************************************************************************/

effsymb (x,y)
int x,y;
 {
  prclcurs (0);
  prcargr (x-2,y,' ');
  prcargr (x+2,y,' ');
  prclcurs (1);
 }



/*****************************************************************************/
/* Recherche de l'existence d'un hétéroatome sur le sommet de coordonnées */
/* (X,Y). S'il est trouvé, la fonction vaut 1 et *pp pointe vers l'atome  */
/* dans le tableau atome et *pn1 et *pn2 sont les 2 lettres du symbole.   */
/* Sinon, la fonction vaut 0 et *pn1 et *pn2 sont inchangés.              */
/*****************************************************************************/

trouvatom (x,y,pp,pn1,pn2)
int x,y;
int *pp;
char *pn1,*pn2;
 {
  int i,si;
  int trouve;

  i = 0;
```

```
 trouve = 0;
 while ((i < atomsuiv) && (trouve == 0))
  {
   si = i;
   if ((atome[i++] == x) && (atome[i++] == y))
    {
     *pp = si;
     *pn1 = atome[i++];
     *pn2 = atome[i];
     trouve = 1;
    }
   i = si + 4;
  }
 return (trouve);
}


/*****************************************************************************/
/* Suppression de atome du sommet de coordonnées (X,Y) dans la table. Si     */
/* tout s'est bien passé, la valeur de la fonction vaut 0, sinon 1.          */
/*****************************************************************************/

suppratom (x,y)
int x,y;
 {
  int i,p;
  char n1,n2;
  int ok;

  ok = 1;
  if (trouvatom (x,y,&p,&n1,&n2) == 1)
   {
    ok = 0;
    i = p + 4;
    while (i < atomsuiv)
      atome[p++] = atome[i++];
    atomsuiv = p;
   }
  return (ok);
 }


/*****************************************************************************/
/* Ajout d'un atome dans la table. Si tout s'est bien passé, la fonction     */
/* vaut 0. Elle vaudra 1 s'il n'y a plus assez de place et 2 si un atome     */
/* existe déjà.                                                              */
/* Contenu du tableau ATOME :                                                */
/*           coordonnées du point concerné                                   */
/*           code ASCII des deux lettres composant le symbole chimique       */
/*****************************************************************************/

addatom (x,y,n1,n2)
int x,y;
char n1,n2;
 {
  int i,p;
  char n3,n4;
  int ok;

  ok = 2;
  if (trouvatom (x,y,&p,&n3,&n4) == 0)
   {
```

```
    ok = 1;
    if ((atomsuiv + 4) < maxatom)
     {
      ok = 0;
      atome[atomsuiv++] = x;
      atome[atomsuiv++] = y;
      atome[atomsuiv++] = n1;
      atome[atomsuiv++] = n2;
     }
   }
  return (ok);
 }



/*****************************************************************************/
/* Supprime un atome à l'écran et y restaure son environnement, c'est-à-  */
/* dire les liaisons simples ou multiples qui y étaient liées. Si l'autre */
/* extrémité des liaisons effacées comporte aussi un atome, il n'est pas  */
/* affecté.                                                                  */
/*****************************************************************************/

restenvat (x,y)
int x,y;
 {
  int liste [12];
  int i,m,p,p2;
  char n1,n2;
  int x2,y2;

  prclcurs (0);
  effsymb (x,y);
  p = 12;
  trouvsomli (x,y,liste,&p);
  i = 0;
  while (i < p)
   {
    x2 = liste[i++];
    y2 = liste[i++];
    line (x,y,x2,y2,1,1);
    line (x,y,x2,y2,2,1);
    if ((m = trouvlm (x,y,x2,y2,&p2)) != 0)
     {
      trml (x,y,x2,y2,1,2);
      if (m == 3)
        trml (x,y,x2,y2,1,3);
     }
    if (trouvatom (x2,y2,&p2,&n1,&n2) == 1)
     prsymb (x2,y2,n1,n2);
   }
  prclcurs (1);
 }



/*****************************************************************************/
/* Retourne une liste de sommets liés au sommet (x,y) par une liaison      */
/* simple ou par une liaison d'un cycle. Chaque sommet retourné est unique */
/* dans la liste.                                                            */
/* *pp doit être égal à 2x le nombre max de sommets à retourner et doit    */
/* être un multiple de 4. En retour, liste contiendra les coordonnées des  */
/* sommets trouvés et *pp leur nombre.                                       */
/*****************************************************************************/
```

```
trouvsomli (x,y,liste,pp)
int x,y;
int liste [];
int *pp;
 {
  int i,si,j,p,n;
  int s1,s2,s3,s4;
  int ok1,ok2,trouve;

  p = 0;
  i = 0;
  while ((i < cyclsuiv) && (p < *pp))
   {
    n = 2 * cycle[i++] + i;
    si = i;
    trouve = 0;
    while ((i < n) && (trouve == 0))
     {
      if ((cycle[i] == x) && (cycle[i+1] == y))
       {
        trouve = 1;
        if ((i + 2) == n)
         {
          s1 = cycle[si];
          s2 = cycle[si+1];
          s3 = cycle[i-2];
          s4 = cycle[i-1];
         }
        else if (i == si)
         {
          s1 = cycle[n-2];
          s2 = cycle[n-1];
          s3 = cycle[i+2];
          s4 = cycle[i+3];
         }
        else
         {
          s1 = cycle[i-2];
          s2 = cycle[i-1];
          s3 = cycle[i+2];
          s4 = cycle[i+3];
         }
        ok1 = 1;
        ok2 = 1;
        j = 0;
        while (j < p)
         {
          if ((ok1 == 1) && (liste[j] == s1) && (liste[j+1] == s2))
            ok1 = 0;
          if ((ok2 == 1) && (liste[j] == s3) && (liste[j+1] == s4))
            ok2 = 0;
          j += 2;
         }
        if (ok1 == 1)
         {
          liste[p++] = s1;
          liste[p++] = s2;
         }
        if (ok2 == 1)
         {
          liste[p++] = s3;
          liste[p++] = s4;
         }
       }
```

```
      i += 2;
     }
    i = n;
   }

  i = 0;
  while ((i < somsuiv) && (p < *pp))
   {
    if ((sommet[i] == x) && (sommet[i+1] == y))
     {
      liste[p++] = sommet[i+2];
      liste[p++] = sommet[i+3];
     }
    else if ((sommet[i+2] == x) && (sommet[i+3] == y))
     {
      liste[p++] = sommet[i];
      liste[p++] = sommet[i+1];
     }
    i += 4;
   }
  *pp = p;
 }



/*************************************************************************/
/* Restauration d'atomes se trouvant sur un sommet participant à une     */
/* liaison qui a été supprimée. Si le sommet n'existe plus, l'atome est  */
/* supprimé à l'écran et dans la table atome. Sinon, il est restauré à   */
/* l'écran.                                                              */
/* liste est la liste des sommets concernés.                             */
/* n est le nombre de sommets.                                           */
/*************************************************************************/

restatls (n,liste)
int liste [];
int n;
 {
  int i,p;
  char n1,n2;
  int x,y;

  i = 0;
  n = 2 * n;
  while (i < n)
   {
    x = liste[i++];
    y = liste[i++];
    if (trouvatom (x,y,&p,&n1,&n2) == 1)
     {
      if (existsom (x,y) != 0)
        prsymb (x,y,n1,n2);
      else
       {
        suppratom (x,y);
        effsymb (x,y);
       }
     }
   }
 }
```

```
/*************************************************************************/
/* Recherche si le sommet (x,y) participe à une liaison ou à un cycle. Si  */
/* c'est le cas, la fonction vaut 1 si le sommet appartient à un cycle, 2  */
/* si le sommet appartient exclusivement à des liaisons et 0 si le sommet  */
/* n'existe pas.                                                           */
/*************************************************************************/

existsom (x,y)
int x,y;
 {
  int i,n;
  int trouve;

  i = 0;
  trouve = 0;
  while ((i < cyclsuiv) && (trouve == 0))
   {
    n = 2 * cycle[i++] + i;
    while ((i < n) && (trouve == 0))
     {
      if ((cycle[i] == x) && (cycle[i+1] == y))
        trouve = 1;
      i += 2;
     }
    i = n;
   }

  i = 0;
  while ((i < somsuiv) && (trouve == 0))
   {
    if ((sommet[i] == x) && (sommet[i+1] == y))
      trouve = 2;
    i += 2;
   }

  return (trouve);
 }



/*************************************************************************/
/* Restauration des atomes, s'il y en a, sur la liste des sommets donnée. */
/*************************************************************************/

restatlp (n,liste)
int liste [];
int n;
 {
  int i,p;
  char n1,n2;
  int x,y;

  i = 0;
  n = n * 2;
  while ( i < n)
   {
    x = liste[i++];
    y = liste[i++];
    if (trouvatom (x,y,&p,&n1,&n2) == 1)
      prsymb (x,y,n1,n2);
   }
 }
```

```
/*****************************************************************************/
/* Translation du dessin d'une molécule à l'écran. Si la molécule occupe     */
/* un seul écran, la molécule peut être déplacée ,n'importe où (pendant      */
/* le déplacement, la molécule est symbolisée par un rectangle). Il existe   */
/* une autre possibilité permettant de dessiner la molécule sur plus d'un    */
/* écran. La partie visible n'est alors qu'une fenêtre sur le dessin. On     */
/* peut s'y déplacer en plaçant le curseur dans en bas ou à gauche de        */
/* l'écran et en cliquant sur un des deux boutons de la souris.              */
/*****************************************************************************/

translat ()
 {
  int xmin,xmax,ymin,ymax;
  int x1,y1,x2,y2;
  int dx,dy;
  int pos [2];
  int sens,i;
  int xpos,ypos;
  char tr;
  static int xdep = 0;
  static int ydep = 0;

  prclcurs (0);
  selcurs (3);
  prclcurs (1);
  tr = 'm';
  while (getposc (pos) == 0)
   {
    if ((pos[1] > 190) && (pos[0] > 9))
     {
      if (tr != 'x')
       {
        xpos = 200-(xdep+320)*5/16;
        prclcurs (0);
        if (tr == 'y')
         {
          line (0,ypos,0,ypos+60,0,1);
          line (1,ypos,1,ypos+60,0,1);
         }
        line (xpos,199,xpos+100,199,1,1);
        line (xpos,198,xpos+100,198,1,1);
        selcurs (4);
        prclcurs (1);
        tr = 'x';
       }
     }
    else if ((pos[0] < 10) && (pos[1] < 191))
     {
      if (tr != 'y')
       {
        ypos = 120-(ydep+200)*3/10;
        prclcurs (0);
        if (tr == 'x')
         {
          line (xpos,199,xpos+100,199,0,1);
          line (xpos,198,xpos+100,198,0,1);
         }
        line (0,ypos,0,ypos+60,1,1);
        line (1,ypos,1,ypos+60,1,1);
        selcurs (5);
        prclcurs (1);
```

```
      tr = 'y';
     }
   }
  else if (tr != 'm')
   {
    prclcurs (0);
    if (tr == 'x')
     {
      line (xpos,199,xpos+100,199,0,1);
      line (xpos,198,xpos+100,198,0,1);
     }
    if (tr == 'y')
     {
      line (0,ypos,0,ypos+60,0,1);
      line (1,ypos,1,ypos+60,0,1);
     }
    tr = 'm';
    selcurs (3);
    prclcurs (1);
   }
 }

for (i = 0; i < 5000; i++);

if ((tr == 'm') && (getposc (pos) != 3))
 {
  minmax (&xmin,&xmax,&ymin,&ymax);
  x1 = xmin;
  y1 = ymax;
  x2 = xmax;
  y2 = ymin;
  if ((x1 >= 0) && (x1 < 320) && (x2 >= 0) && (x2 < 320))
    if ((y1 >= 0) && (y1 < 200) && (y2 >= 0) && (y2 < 200))
     {
      dx = x2 - x1;
      dy = y2 - y1;
      prclcurs (0);
      trabox (x1,y1,x2,y2,1);
      pos[0] = x1;
      pos[1] = y1;
      putposc (pos);
      selcurs (2);
      windcurs (0,x1+319-xmax,y1-ymin,199);
      prclcurs (1);
      while (getposc (pos) != 0);
      while (getposc (pos) == 0)
       if ((pos[0] != x1) || (pos[1] != y1))
        {
         trabox (x1,y1,x2,y2,0);
         x1 = pos[0];
         y1 = pos[1];
         x2 = x1 + dx;
         y2 = y1 + dy;
         trabox (x1,y1,x2,y2,1);
        }
      trabox (x1,y1,x2,y2,0);
      for (i = 0; i < 5000; i++);
      if (getposc (pos) == 1)
       {
        dx = x1 - xmin;
        dy = y1 - ymax;
        transcoord (dx,dy);
        prclcurs (0);
        setgrs ();
```

```
        restmolec ();
        prclcurs (1);
       }
     }
  }

 if (tr == 'x')
  {
   sens = getposc (pos);
  if (sens == 1)
   {
    dx = -pos[0];
    if ((xdep + dx) <= -320)
      dx = -320 - xdep;
   }
  else if (sens == 2)
   {
    dx = pos[0];
    if ((xdep + dx) > 320)
      dx = 320 - xdep;
   }
  if (sens != 3)
   {
    xdep += dx;
    transcoord (dx,0);
    prclcurs (0);
    setgrs ();
    restmolec ();
    prclcurs (1);
    }
  }

 if (tr == 'y')
  {
   sens = getposc (pos);
   if (sens == 1)
    {
     dy = -pos[1];
     if ((ydep + dy) <= -200)
       dy = -200 - ydep;
    }
   else if (sens == 2)
    {
     dy = pos[1];
     if ((ydep + dy) > 200)
       dy = 200 - ydep;
    }
   if (sens != 3)
    {
     ydep += dy;
     transcoord (0,dy);
     prclcurs (0);
     setgrs ();
     restmolec ();
     prclcurs (1);
    }
  }
}
```

```
/*****************************************************************************/
/* Tracé d'un rectangle dont le sommet à gauche le plus bas est (x1, y1) et */
/* le sommet à droite le plus haut est (x2,y2). Le paramaètre p est le même */
/* que le deuxième de line.                                                  */
/*****************************************************************************/

trabox (x1,y1,x2,y2,p)
int x1,y1,x2,y2;
int p;
 {
  prclcurs (0);
  line (x1,y1,x2,y1,p,0);
  line (x2,y1,x2,y2,p,0);
  line (x2,y2,x1,y2,p,0);
  line (x1,y2,x1,y1,p,0);
  prclcurs (1);
 }



/*****************************************************************************/
/* Recherche les valeurs extrémales des coordonnées des sommets du dessin  */
/* de la molécule.                                                           */
/*****************************************************************************/

minmax (pxmin,pxmax,pymin,pymax)
int *pxmin,*pxmax,*pymin,*pymax;
 {
  int xi,xa,yi,ya;
  int i,n;

  xi = 320;
  xa = 0;
  yi = 199;
  ya = 0;
  i = 0;
  while (i < somsuiv)
   {
    if (sommet[i] > xa)
      xa = sommet[i];
    if (sommet[i] < xi)
      xi = sommet[i];
    if (sommet[i+1] > ya)
      ya = sommet[i+1];
    if (sommet[i+1] < yi)
      yi = sommet[i+1];
    i += 2;
   }
  i = 0;
  while (i < cyclsuiv)
   {
    n = 2 * cycle[i++] + i;
    while (i < n)
     {
      if (cycle[i] > xa)
        xa = cycle[i];
      if (cycle[i] < xi)
        xi = cycle[i];
      if (cycle[i+1] > ya)
        ya = cycle[i+1];
      if (cycle[i+1] < yi)
        yi = cycle[i+1];
```

```
      i += 2;
     }
   }
  if (xi == 320)
   {
    yi = xi = 0;
    xa = 319;
    ya = 199;
   }
  *pxmin = xi;
  *pxmax = xa;
  *pymin = yi;
  *pymax = ya;
 }



/*************************************************************************/
/* Réalise la translation des coordonnées de tous les sommets en fonction */
/* de ex et ey.                                                           */
/*************************************************************************/

transcoord (ex,ey)
int ex,ey;
 {
  int i,n;

  i = 0;
  while (i < somsuiv)
   {
    sommet[i] = sommet[i++] + ex;
    sommet[i] = sommet[i++] + ey;
   }
  i = 0;
  while (i < cyclsuiv)
   {
    n = 2 * cycle[i++] + i;
    while (i < n)
     {
      cycle[i] = cycle[i++] + ex;
      cycle[i] = cycle[i++] + ey;
     }
   }
  i = 0;
  while (i < lmsuiv)
   {
    n = i + 5;
    i++;
    while (i < n)
     {
      lmsommet[i] = lmsommet[i++] + ex;
      lmsommet[i] = lmsommet[i++] + ey;
     }
   }
  i = 0;
  while (i < atomsuiv)
   {
    atome[i] = atome[i++] + ex;
    atome[i] = atome[i++] + ey;
    i += 2;
   }
 }
```

```
/*****************************************************************************/
/* On redessine la molécule dont les composantes sont sauvegardées dans  */
/* les tables.                                                               */
/*****************************************************************************/

restmolec ()
 {
  restcycle ();
  restls ();
  restlm ();
  restat ();
 }



restcycle ()
 {
  int i,j,n,n2;
  int lx [tmcycle],ly [tmcycle];

  i = 0;
  while (i < cyclsuiv)
   {
    n = cycle[i++];
    n2 = 2 * n + i;
    j = 0;
    while (i < n2)
     {
      lx[j] = cycle[i++];
      ly[j++] = cycle[i++];
     }
    tracycle (n,lx,ly,1,0);
    tracycle (n,lx,ly,2,0);
   }
 }




restls ()
 {
  int i;
  int x0,y0,x1,y1;

  i = 0;
  while (i < somsuiv)
   {
    x0 = sommet [i++];
    y0 = sommet [i++];
    x1 = sommet [i++];
    y1 = sommet [i++];
    line (x0,y0,x1,y1,1,1);
    line (x0,y0,x1,y1,2,1);
   }
 }




restlm ()
 {
  int i,m;
  int x0,y0,x1,y1;
```

```
  i = 0;
  while (i < lmsuiv)
   {
    m = lmsommet[i++];
    x0 = lmsommet[i++];
    y0 = lmsommet[i++];
    x1 = lmsommet[i++];
    y1 = lmsommet[i++];
    trml (x0,y0,x1,y1,1,2);
    if (m == 3)
      trml (x0,y0,x1,y1,1,3);
   }
 }



restat ()
 {
  int i;
  int x,y;
  char n1,n2;

  i = 0;
  while (i < atomsuiv)
   {
    x = atome[i++];
    y = atome[i++];
    n1 = atome[i++];
    n2 = atome[i++];
    prsymb (x,y,n1,n2);
   }
 }



/*************************************************************************/
/* Initialisation de la table contenant les valeurs d'arc tangente. S'il  */
/* existe un fichier avec ces données et pour la même valeur de dimalpha, */
/* celui-ci sera utilisé pour initialiser le tableau sinon toutes les     */
/* valeurs seront calculées et un essai de sauvegarde sera effectué.      */
/*************************************************************************/

initalpha ()
 {
  int x,y;
  int fin,n;
  double atan (),x1, y1;
  int file,length,mode,status;

  printf ("Initialisation du tableau ALPHA \n");
  fin = dimalpha;

  mode = O_RDONLY | O_RAW;
  length = fin * fin * 2;
  file = open ("TANGENTE.TBL",mode);
  if ((file != -1) && (read (file,&n,2) != -1) && (n == length))
   {
    printf ("     Lecture des données sur disque \n");
    status = read (file,alpha,length);
    if (status != length)
     {
      printf ("     La lecture des données a échoué ! \n");
```

```
      status = -1;
     }
   }
  else
    status = -1;
  close (file);

  if (status == -1)
   {
    printf ("      Calcul des valeurs du tableau \n");
    for (x = 1; x < (fin-1); x++)
      for (y = x+1; y < fin; y++)
       {
        x1 = x;
        y1 = y;
        alpha [x] [y] = atan (x1 / y1) / 3.141593 * 180.0 + 0.5;
        alpha [y] [x] = 90 - alpha [x] [y];
       }
    for (x = 0; x < fin; x++)
     {
      alpha [0] [x] = 0;
      alpha [x] [0] = 90;
      alpha [x] [x] = 45;
     }
    mode = O_WRONLY | O_CREAT | O_RAW;
    file = open ("TANGENTE.TBL",mode);
    if (file != -1)
     {
      printf ("      Sauvegarde des données sur disque \n");
      status = write (file,&length,2);
      if ((status == 2) && (write (file,alpha,length) == length))
        printf ("      Sauvegarde terminée ! \n");
      else
       {
        status = -1;
        printf ("      La sauvegarde a échoué ! \n");
       }
     }
    else
     {
      printf ("      La sauvegarde des données ne peut être faite ! \n");
      status = -1;
     }
    close (file);
    if (status == -1)
      unlink ("TANGENTE.TBL");
   }
 }



/*************************************************************************/
/* Recherche de l'angle d'une liaison dont les sommets sont (x0,y0) et   */
/* (x1,y1).                                                              */
/*************************************************************************/

trouvalpha (x0,y0,x1,y1)
int x0,y0,x1,y1;
 {
  int dx,dy;
  int sx,sy;
  int angle;

  dx = x1 - x0;
```

```
  dy = y1 - y0;
  sx = (dx < 0) ? -1 : 1;
  sy = (dy < 0) ? -1 : 1;
  dx = dx * sx;
  dy = dy * sy;
  angle = alpha [dx] [dy];
  if ((sy < 0) && (sx > 0))
    angle = 180 - angle;
  if ((sy < 0) && (sx < 0))
    angle = angle + 180;
  if ((sy > 0) && (sx < 0))
    angle = 360 - angle;
  return (angle);
 }



/*************************************************************************/
/* Initialisation des valeurs de sin et cos pour des angles entre 0 et   */
/* 360 degrés.                                                           */
/*************************************************************************/

initvlsc ()
 {
  int i;
  double sin (), cos (), x, y;
  int file,length,mode,status;

  printf ("Initialisation des tables VLSIN et VLCOS \n");
  mode = O_RDONLY | O_RAW;
  length = 1440;
  file = open ("VLSINCOS.TBL",mode);
  if (file != -1)
   {
    printf ("     Lecture des données sur disque \n");
    status = read (file,vlsin,length);
    if (status != length)
     {
      printf ("     La lecture des données a échoué ! \n");
      status = -1;
     }
   }
  else
    status = -1;
  close (file);

  if (status == -1)
   {
    status = 0;
    printf ("     Calcul des valeurs des tables \n");
    for (i = 1; i < 90; i++)
     {
      x = i * 3.141593 / 180;
      y = sin (x) * 11;
      y = (y < 0) ? y - 0.5 : y + 0.5;
      vlsin [180-i] = vlsin [i] = y;
      vlsin [180+i] = vlsin [360-i] = -y;
      y = cos (x) * 11;
      y = (y < 0) ? y - 0.5 : y + 0.5;
      vlcos [360-i] = vlcos [i] = y;
      vlcos [180+i] = vlcos [180-i] = -y;
     }
    vlsin [0] = vlsin [180] = vlcos [90] = vlcos [270] = 0;
    vlsin [90] = vlcos [0] = 11;
```

```
    vlsin [270] = vlcos [180] = -11;

    mode = O_WRONLY | O_CREAT | O_RAW;
    file = open ("VLSINCOS.TBL",mode);
    if (file != -1)
     {
      printf ("     Sauvegarde des données sur disque \n");
      if (write (file,vlsin,length) == length)
        printf ("     Sauvegarde terminée ! \n");
      else
       {
        status = -1;
        printf ("     La sauvegarde a échoué ! \n");
       }
     }
    else
     {
      printf ("     La sauvegarde des données ne peut être faite ! \n");
      status = -1;
     }
    close (file);
    if (status == -1)
      unlink ("VLSINCOS.TBL");
   }
}



/*************************************************************************/
/* Sauvegarde des coordonnées des sommets de la molécule sur le disque  */
/* virtuel.                                                              */
/*************************************************************************/

savemolec ()
 {
  int mode,file,length;

  mode = O_WRONLY | O_CREAT | O_RAW;
  length = (maxatom + maxlm + maxsom + maxcycle) * 2 + 8;
  file = open ("C:COORDMOL.$$$",mode);
  write (file,atome,length);
  close (file);
 }
```

```
#include <fcntl.h>
#define maxsom 1000
#define maxcycle 1000
#define maxlm 1000
#define maxatom 1000
#define maxcoup 1000        /* place max pour les couples de cycles */
#define maxcc 1000          /* place max pour les couples de couples */

int atome [maxatom];
int atomsuiv;

int lmsommet [maxlm];
int lmsuiv;

int sommet [maxsom];
int somsuiv;

int cycle [maxcycle];
int cyclsuiv;
int numcycl;
int deridcy = 0;            /* dernier identifiant utilisé pour les cycles */

int isolcycl [maxcycle/5];

int listcoup [maxcoup];
int coupsuiv = 0;

int tbl [maxcycle];
int nbl;

int idcycle [maxcc/5*3];
int tailcyc [maxcc/5*3];
int distlicy [maxcc/5*3];
int tripcsuiv = 0;

int listchai [maxsom/4*3];
int chaisuiv = 0;

int listid [(maxcycle+maxsom)/2*3];
int suivid = 0;
int idcour = 0;

int listcc [maxcc];
int ccsuiv = 0;

int numli [(maxcycle+maxsom)/2];

int listliai [maxcycle+maxsom];
int liaisuiv = 0;

main ()
 {
  int i,j,n;
  int tc1,tc2,tc3;
  int nc1,nc2,nc3;
  int l1,l2;
  int d;
  int isolcoup [maxcoup/4];
  int erreur;

  loadmol ();
  erreur = relacoup ();
  if (erreur == 1)
    printf ("\n Ensemble de cycles trop complexe ! \n\n");
```

```
else
 {
  i = 0;
  j = 0;
  printf ("\n    Description des cycles \n");
  printf   ("    ---------------------- \n");
  while (i < cyclsuiv)
   {
    n = cycle[i++];
    if (isolcycl[j++] == 1)
      printf ("* ");
    else
      printf ("  ");
    printf ("C%1d(%2d) : ",n,cycle[i++]);
    n += i;
    while (i < n)
      printf ("%4d ",cycle[i++]);
    printf ("\n");
   }
  printf ("\n");

  i = j = 0;
  while (i < nbl)
   {
    n = tbl[i++];
    if (n == 1)
      isolcoup[j++] = 1;
    else
     {
      n = j + n;
      while (j < n)
        isolcoup[j++] = 0;
     }
   }
  i = j = 0;
  printf ("\n    Description des couples de cycles \n");
  printf   ("    --------------------------------- \n");
  while (i < coupsuiv)
   {
    if (isolcoup[j++] == 1)
      printf ("* ");
    else
      printf ("  ");
    printf ("C%1d(%2d)",cycle[listcoup[i]],cycle[listcoup[i++] + 1]);
    printf (" - C%1d(%2d)",cycle[listcoup[i]],cycle[listcoup[i++] + 1]);
    printf (" : %4d",cycle[listcoup[i++]]);
    if (listcoup[i++] != -1)
      printf (",%4d",cycle[listcoup[i-1]]);
    printf ("\n");
   }
  printf ("\n");

  printf ("\n    Description finale des cycles \n");
  printf   ("    ----------------------------- \n");
  i = 0;
  while (i < tripcsuiv)
   {
    if (idcycle[i] == -1)
     {
      printf ("                                ****** \n");
      i += 3;
     }
    else
     {
```

```
        tc1 = tailcyc[i];
        nc1 = idcycle[i];
        d = distlicy[i++];
        tc2 = tailcyc[i];
        nc2 = idcycle[i];
        l1 = distlicy[i++];
        tc3 = tailcyc[i];
        nc3 = idcycle[i];
        l2 = distlicy[i++];
        printf ("  C%1d(%2d) - C%1d(%2d) - C%1d(%2d) ",tc1,nc1,tc2,nc2,tc3,nc3);
        printf ("  distance : %1d   liaisons : %1d, %1d \n",d,l1,l2);
       }
     }
   }

  findch ();
  printf ("\n    Description des chaines \n");
  printf   ("    ---------------------- \n");
  i = 0;
  while (i < chaisuiv)
   {
    if (listchai[i] == -1)
      printf ("                              ****** \n");
    else
     {
      printf ("  premier atome :%4d ",listchai[i+1]);
      printf (" -  longueur : %2d ",listchai[i]);
      printf (" -  dernier atome :%4d \n",listchai[i+2]);
     }
    i += 3;
   }
  savdescr ();
  exit (0);
 }



/*************************************************************************/
/* Lecture de la description en coordonnées écran d'une molécule sur le  */
/* disque virtuel.                                                       */
/*************************************************************************/

loadmol ()
 {
  int mode,file,length;

  length = (maxatom + maxlm + maxsom + maxcycle) * 2 + 8;
  mode = O_RDONLY | O_RAW;
  file = open ("C:COORDMOL.$$$",mode);
  read (file,atome,length);
  close (file);
 }



/*************************************************************************/
/* Sauvegarde de la nouvelle description de la molécule sur le disque    */
/* virtuel.                                                              */
/*************************************************************************/

savdescr ()
 {
```

```
  int mode,file,length;

  length = (maxcycle/5*11 + maxcoup + maxcc/5*9 + maxsom/4*3)*2 + 14;
  mode = O_WRONLY | O_CREAT | O_RAW;
  file = open ("C:DESCRMOL.$$$",mode);
  write (file,cycle,length);
 }



/*************************************************************************/
/* Conversion des coordonnées des sommets des cycles en numéros de sommets. */
/* Le tableau CYCLE garde la même structure mais avec cette modification et */
/* on ajoute une caractéristique supplémentaire à chaque cycle qui est un   */
/* numéro identifiant.                                                      */
/* On a donc : taille cycle, identifiant cycle, identifiants des sommets.   */
/*************************************************************************/

convcycle ()
 {
  int i,j,jj,n;
  int x,y;
  int ids,idc;

  idc = 1;
  i = j = 0;
  while (i < cyclsuiv)
   {
    n = cycle[i++];
    cycle[j++] = n;
    jj = j++;
    n = 2 * n + i;
    while (i < n)
     {
      x = cycle[i++];
      y = cycle[i++];
      if ((ids = convcn (x,y)) != 0)
        cycle[j++] = ids;
      else
        cycle[j++] = addids (x,y);
     }
    cycle[jj] = idc++;
   }
  cyclsuiv = j;
  numcycl = --idc;
 }



/*************************************************************************/
/* Recherche si le sommet de coordonnées (x,y) a déjà été converti. Si   */
/* c'est le cas, la fonction retourne son numéro identifiant.            */
/*************************************************************************/

convcn (x,y)
int x,y;
 {
  int i,numid;

  numid = 0;
  i = 0;
  while (i < suivid)
   {
```

```
    if ((x == listid[i]) && (y == listid[i+1]))
     {
      numid = listid[i+2];
      i = suivid;
     }
    i += 3;
   }
  return (numid);
 }



/*************************************************************************/
/* Ajoute dans la table un sommet de coordonnées (x,y) et lui attribue un  */
/* numéro identifiant. Ce numéro est retourné par la fonction. Pour conser- */
/* ver la cohérence de la table, il faut d'abord vérifer si le sommet n'a  */
/* déjà pas été converti (avec convcn).                                    */
/*************************************************************************/

addids (x,y)
int x,y;
 {
  listid[suivid++] = x;
  listid[suivid++] = y;
  listid[suivid++] = ++idcour;
  return (idcour);
 }



/*************************************************************************/
/* Recherche de toutes les paires de cycles dans une molécule (les sommets  */
/* des cycles doivent avoir été convertis). Elles sont mémorisées dans une  */
/* table LISTCOUP sous la forme :                                          */
/*       pointeur (dans la table CYCLE) vers le premier cycle              */
/*       pointeur (dans la table CYCLE) vers le deuxième cycle             */
/*       pointeur (dans la table CYCLE) vers le sommet 1 de la liaison     */
/*       pointeur (dans la table CYCLE) vers le sommet 2 de la liaison     */
/* Les sommets de la liaison (commune aux deux cycles) sont décrits dans   */
/* l'ordre (antihorlogique) d'apparition dans le premier cycle.            */
/* Dans le cas des dérivés spiraniques, les cycles ne sont liés que par un  */
/* carbone. Dans ce cas, le dernier pointeur vaut -1.                      */
/* De plus, pour chaque cycle, on définit dans la table ISOLCYCL s'il par- */
/* ticipe ou non à au moins une paire. Si le cycle est isolé, alors ISOLCYCL*/
/* vaut 1 pour ce cycle, sinon 0 (La valeur pour le premier cycle est dans  */
/* ISOLCYCL [0] ).                                                         */
/*************************************************************************/

paircycl ()
 {
  int i,j,jj,si,sj,f1,f2,som;
  int k1,k2;
  int n1,n2,id1,id2,ptr1,ptr2;
  int ok,inv,erreur;

  i = 0;
  while (i < numcycl)
    isolcycl[i++] = 1;
  k1 = -1;
  i = 0;
  while ((i < cyclsuiv) && (coupsuiv < maxcoup))
   {
    k2 = ++k1;
    ptr1 = i;
```

```
n1 = cycle[i++];
id1 = cycle[i++];
j = i + n1;
f1 = j;
si = i;
while (j < cyclsuiv)
 {
  k2++;
  i = si;
  ptr2 = j;
  n2 = cycle[j++];
  id2 = cycle[j++];
  f2 = j + n2;
  sj = j;
  ok = 1;
  while ((i < f1-1) && (ok == 1))
   {
    j = sj;
    while ((j < f2) && (ok == 1))
      if (cycle[j] == cycle[i])
        ok = 0;
      else
        j++;
    i++;
   }
  if (ok == 0)
   {
    isolcycl[k1] = 0;
    isolcycl[k2] = 0;
    i--;
    inv = 0;
    som = i;
    if (i == si)
     {
      i++;
      jj = j;
      j--;
      if (j < sj)
        j = f2 - 1;
      if (cycle[i] == cycle[j])
        ok = 1;
      else
       {
        i = f1 - 1;
        j = jj + 1;
        if (j == f2)
          j = sj;
        inv = 1;
        if (cycle[i] == cycle[j])
          ok = 1;
       }
     }
    else
     {
      i++;
      j--;
      if (j < sj)
        j = f2 - 1;
      if (cycle[i] == cycle[j])
        ok = 1;
     }
    if (ok == 1)
     {
      listcoup[coupsuiv++] = ptr1;
```

```
        listcoup[coupsuiv++] = ptr2;
        if (inv == 0)
         {
          listcoup[coupsuiv++] = som;
          listcoup[coupsuiv++] = i;
         }
        else
         {
          listcoup[coupsuiv++] = i;
          listcoup[coupsuiv++] = som;
         }
       }
      else
       {
        listcoup[coupsuiv++] = ptr1;
        listcoup[coupsuiv++] = ptr2;
        listcoup[coupsuiv++] = som;
        listcoup[coupsuiv++] = -1;
       }
     }
    j = f2;
   }
  i = f1;
 }
 if (coupsuiv >= maxcoup)
   erreur = 1;
 else
   erreur = 0;
 return (erreur);
}



/*****************************************************************************/
/* Regroupement des paires de cycles de façon à séparer les ensembles de     */
/* cycles indépendants. Le nombre d'ensembles détectés sera NBL et la table  */
/* TBL reprendra le nombre de paires de cycles pour chacun des ensembles.    */
/*****************************************************************************/

tripair ()
 {
  int ptrsuiv,ptrcour,ptr1,i;
  int cy1,cy2,som1,som2;
  int trouve;

  nbl = 0;
  ptrsuiv = ptrcour = 4;
  ptr1 = 0;
  while (ptrsuiv < coupsuiv)
   {
    while (ptrcour < coupsuiv)
     {
      cy1 = listcoup[ptrcour];
      cy2 = listcoup[ptrcour+1];
      i = ptr1;
      trouve = 0;
      while ((i < ptrsuiv) && (trouve == 0))
        if ((cy1 == listcoup[i]) || (cy1 == listcoup[i+1]))
          trouve = 1;
        else if ((cy2 == listcoup[i]) || (cy2 == listcoup[i+1]))
          trouve = 1;
        else
          i += 4;
```

```
    if ((trouve == 1) && (ptrcour != ptrsuiv))
     {
      listcoup[ptrcour] = listcoup[ptrsuiv];
      listcoup[ptrcour+1] = listcoup[ptrsuiv+1];
      som1 = listcoup[ptrcour+2];
      som2 = listcoup[ptrcour+3];
      listcoup[ptrcour+2] = listcoup[ptrsuiv+2];
      listcoup[ptrcour+3] = listcoup[ptrsuiv+3];
      listcoup[ptrsuiv++] = cy1;
      listcoup[ptrsuiv++] = cy2;
      listcoup[ptrsuiv++] = som1;
      listcoup[ptrsuiv++] = som2;
     }
    else if (trouve == 1)
      ptrsuiv += 4;
    ptrcour += 4;
   }
  tbl[nbl++] = (ptrsuiv - ptr1) / 4;
  ptr1 = ptrsuiv;
  ptrsuiv += 4;
  ptrcour = ptrsuiv;
 }
 if (ptrcour-4 < coupsuiv)
  tbl[nbl++] = 1;
}



/*****************************************************************************/
/* Recherche des relations entre les couples de cycles. Elles sont établies */
/* pour toutes paires de couples de cycles ayant un cycle en commun. Elles  */
/* sont mémorisées dans la table LISTCC.                                    */
/* Description de la table :                                                */
/*   pointeur vers le premier couple                                        */
/*   pointeur vers le deuxième couple                                       */
/*   numéro du cycle commun dans le premier couple                          */
/*   numéro du cycle commun dans le premier couple                          */
/*   nombre de liaisons sur le cycle commun séparant les deux autres cycles */
/* On obtient ainsi une relation entre 3 cycles de la molécule.             */
/* L'ordre des deux premiers pointeurs est tel que en partant de la liaison */
/* des deux cycles du premier couple et en parcourant ensuite le cycle com- */
/* mun dans le sens antihorlogique, on arrive à la liaison du deuxième      */
/* couple en ayant parcouru le minimum de liaison sur le cycle commun (qui  */
/* est "le nombre de liaisons séparant les deux autres cycles").            */
/* Les relations entre les paires sont établies indépendamment pour chaque  */
/* ensemble de paires défini par TRIPAIR. Ces ensembles de relations seront */
/* séparés par un enregistrement d'une relation dont toutes les valeurs     */
/* seront à -1.                                                             */
/*****************************************************************************/

relacoup ()
 {
  int i,j,k,n;
  int erreur;

  convcycle ();
  erreur = paircycl ();
  if (erreur == 0)
   {
    tripair ();
    n = k = 0;
    while ((n < nbl) && (ccsuiv < maxcc))
     {
      i = k;
```

```
    k = k + 4*tbl[n++];
    if (i+4 < k)
     {
      while ((i < k) && (ccsuiv < maxcc))
       {
        j = i + 4;
        while ((j < k) && (ccsuiv < maxcc))
         {
          if (listcoup[i] == listcoup[j])
            createc (i,j,1,1);
          else if (listcoup[i+1] == listcoup[j])
            createc (i,j,2,1);
          else if (listcoup[i] == listcoup[j+1])
            createc (i,j,1,2);
          else if (listcoup[i+1] == listcoup[j+1])
            createc (i,j,2,2);
          j += 4;
         }
        i += 4;
       }
      listcc[ccsuiv++] = -1;
      listcc[ccsuiv++] = -1;
      listcc[ccsuiv++] = -1;
      listcc[ccsuiv++] = -1;
      listcc[ccsuiv++] = -1;
     }
    }
   }
  if ((ccsuiv >= maxcc) || (erreur == 1))
    erreur = 1;
  else
   {
    erreur = 0;
    convrela ();
   }
  return (erreur);
 }



/*****************************************************************************/
/* Détermination effective des éléments de la table LISTCC                   */
/*****************************************************************************/

createc (i,j,p1,p2)
int i,j;
int p1,p2;
 {
  int s1,s2;
  int pcc,fcc,k;
  int d,tc;
  int t1,t2;
  int inv;

  s1 = cycle[listcoup[i+4-p1]];
  s2 = cycle[listcoup[j+1+p2]];
  t1 = 1;
  t2 = 2;
  if ((listcoup[i+3] != -1) && (listcoup[j+3] == -1))
   {
    s2 = cycle[listcoup[j+2]];
    t2 = 1;
   }
  else if ((listcoup[i+3] == -1) && (listcoup[j+3] != -1))
```

```
  {
   s1 = cycle[listcoup[i+2]];
   t2 = 1;
  }
 else if ((listcoup[i+3] == -1) && (listcoup[j+3] == -1))
  {
   s1 = cycle[listcoup[i+2]];
   s2 = cycle[listcoup[j+2]];
   t1 = 0;
   t2 = 0;
  }
 inv = 0;
 pcc = listcoup[i+p1-1];
 tc = cycle[pcc];
 pcc += 2;
 fcc = pcc + tc;
 k = pcc;
 while (cycle[k++] != s1);
 k--;
 d = 0;
 while (cycle[k++] != s2)
  {
   d++;
   if (k == fcc)
     k = pcc;
  }
 if (d > ((tc-t1)/2))
  {
   inv = 1;
   d = tc - t2 - d;
  }
 if (inv == 0)
  {
   listcc[ccsuiv++] = i;
   listcc[ccsuiv++] = j;
   listcc[ccsuiv++] = p1;
   listcc[ccsuiv++] = p2;
  }
 else
  {
   listcc[ccsuiv++] = j;
   listcc[ccsuiv++] = i;
   listcc[ccsuiv++] = p2;
   listcc[ccsuiv++] = p1;
  }
 listcc[ccsuiv++] = d;
}
```

```
/*****************************************************************************/
/* Conversion de la table listcc produite par RELACOUP en 3 tables dont les */
/* descriptions sont les suivantes :                                         */
/* IDCYCLE reprend dans l'ordre le numéro identifiant des 3 cycles.          */
/* TAILCYC reprend la taille des 3 cycles.                                   */
/* DISTLICY  distance entre le premier et dernier cycle.                     */
/*           nbre d'atomes dans la liaison entre les 2 premiers cycles       */
/*           nbre d'atomes dans la liaison entre les 2 derniers cycles       */
/*****************************************************************************/

convrela ()
 {
  int i;
```

```
  int nc1,nc2,nc3,tc1,tc2,tc3,d,l1,l2;

  i = 0;
  tripcsuiv = 0;
  while (i < ccsuiv)
   {
    if (listcc[i] == -1)
     {
      idcycle[tripcsuiv] = tailcyc[tripcsuiv] = distlicy[tripcsuiv++] = -1;
      idcycle[tripcsuiv] = tailcyc[tripcsuiv] = distlicy[tripcsuiv++] = -1;
      idcycle[tripcsuiv] = tailcyc[tripcsuiv] = distlicy[tripcsuiv++] = -1;
     }
    else
     {
      tc1 = cycle[listcoup[listcc[i] + 1 - listcc[i+2] + 1]];
      nc1 = cycle[listcoup[listcc[i] + 1 - listcc[i+2] + 1] + 1];
      if (listcoup[listcc[i] + 3] == -1)
        l1 = 1;
      else
        l1 = 2;
      tc2 = cycle[listcoup[listcc[i] + listcc[i+2] - 1]];
      nc2 = cycle[listcoup[listcc[i] + listcc[i+2] - 1] + 1];
      tc3 = cycle[listcoup[listcc[i+1] + 1 - listcc[i+3] + 1]];
      nc3 = cycle[listcoup[listcc[i+1] + 1 - listcc[i+3] + 1] + 1];
      if (listcoup[listcc[i+1] + 3] == -1)
        l2 = 1;
      else
        l2 = 2;
      d = listcc[i+4];
      idcycle[tripcsuiv] = nc1;
      tailcyc[tripcsuiv] = tc1;
      distlicy[tripcsuiv++] = d;
      idcycle[tripcsuiv] = nc2;
      tailcyc[tripcsuiv] = tc2;
      distlicy[tripcsuiv++] = l1;
      idcycle[tripcsuiv] = nc3;
      tailcyc[tripcsuiv] = tc3;
      distlicy[tripcsuiv++] = l2;
     }
    i += 5;
   }
 }




/*************************************************************************/
/* Conversion des coordonnées des sommets des liaisons simples en un numéro */
/* identifiant. La table SOMMET à la structure suivante après conversion :  */
/*          numéro premier sommet                                          */
/*          numéro deuxième sommet                                         */
/* De plus, le dernier numéro de sommet utilisé par les cycles est sauvé.   */
/*************************************************************************/

convsommet ()
 {
  int i,j;
  int x,y;
  int ids;

  deridcy = idcour;
  i = j = 0;
  while (i < somsuiv)
```

```
  {
   x = sommet[i++];
   y = sommet[i++];
   if ((ids=convcn(x,y)) != 0)
     sommet[j++] = ids;
   else
     sommet[j++] = addids(x,y);
  }
 somsuiv = j;
}



/*************************************************************************/
/* Classement de la liste des liaisons sur le premier sommet. Chaque liai- */
/* son est deux fois dans liste, une fois dans l'ordre sommet 1, sommet 2  */
/* et une fois dans l'autre sens.                                          */
/* De plus, NUMLI[numéro sommet] contiendra le nombre de liaisons auxquels */
/* participent le sommet dont le numéro est donné.                         */
/*************************************************************************/

trisom ()
 {
  int i,j,k,fin;
  int ok;
  int somint1,somint2;

  j = somsuiv;
  for (i = 0; i < somsuiv; i += 2)
   {
    sommet[j++] = sommet[i+1];
    sommet[j++] = sommet[i];
   }
  ok = 0;
  fin = 2 * somsuiv - 2;
  while (ok == 0)
   {
    ok = 1;
    for (i=0; i<fin; i+=2)
      if (sommet[i] > sommet[i+2])
       {
        ok = 0;
        somint1 = sommet[i];
        somint2 = sommet[i+1];
        sommet[i] = sommet[i+2];
        sommet[i+1] = sommet[i+3];
        sommet[i+2] = somint1;
        sommet[i+3] = somint2;
       }
   }
  i = 0;
  fin = somsuiv * 2;
  while (i < fin)
   {
    somint1 = sommet[i];
    j = 1;
    i += 2;
    while ((sommet[i] == somint1) && (i < fin))
     {
      i += 2;
      j++;
     }
    numli[somint1] = j;
   }
```

```
}



/*************************************************************************/
/* Copie des liaisons dans la table LISTLIAI                             */
/*************************************************************************/

copyls ()
 {
  int i,fin;

  i = 0;
  fin = 2 * somsuiv;
  while (i < fin)
    listliai[i] = sommet[i++];
  liaisuiv = fin;
 }


/*************************************************************************/
/* Correction des valeurs de NUMLI en fonction des sommets utilisés par les */
/* cycles. Si NUMLI[num sommet] == 2 pour ce sommet et qu'il intervient  */
/* aussi dans un cycle, alors NUMLI[num sommet] = 3. Cela permettra de   */
/* détecter effectivement deux chaînes plutôt qu'une dans le cas de figures */
/* comme celle-ci :      _          on aura deux chaînes de 2 atomes     */
/*                      <_><        plutot qu'une de 3 atomes            */
/*************************************************************************/

corrcycl ()
 {
  int i,fin;
  int som;

  if (deridcy != 0)
   {
    i = 0;
    fin = 2 * somsuiv;
    while (i < somsuiv)
     {
      if (((som = sommet[i]) <= deridcy) && (numli[som] == 2))
        numli[som] = 3;
      i += 2;
     }
   }
 }



/*************************************************************************/
/* Recherche de la première liaison rencontrée contenant un sommet donné. */
/* La recherche ne se fait que sur le premier sommet des liaisons car    */
/* celles-ci sont présentes dans les deux sens. Si le sommet est trouvé, on */
/* rend un pointeur dans SOMMET vers la première liaison encore existente */
/* (c'est-à-dire une liaison dont le deuxième sommet est différent de 0) */
/* auquel ce sommet participe. Si aucune liaison n'est trouvée, la valeur */
/* retournée est 2*SOMSUIV.                                              */
/*************************************************************************/

findli (som)
int som;
```

```
{
 int p,q,k;
 int fin;
 int present;

 p = 0;
 q = 2*somsuiv-1;
 present = 0;
 while ((present == 0) && (p <= q))
  {
   k = (p + q) / 2 & 65534;
   if (som == sommet[k])
     present = 1;
   else if (som > sommet[k])
     p = k + 2;
   else
     q = k - 2;
  }
 fin = 2 * somsuiv;
 if (present == 1)
  {
   while ((k != 0) && (sommet[k] == som))
     k -= 2;
   if (sommet[k] != som)
     k += 2;
   while ((sommet[k] == som) && (sommet[k+1] == 0) && (k < fin))
     k += 2;
   if ((sommet[k] != som) || (k >= fin))     /* normalement impossible dans */
     k = fin;                                /* les conditions d'appel.     */
  }
 else
   k = fin;
 return (k);
}



/*************************************************************************/
/* Recherche du premier sommet rencontré n'appartenant plus qu'à une seule  */
/* liaison. Le numéro du deuxième sommet de la liaison sera différent de 0. */
/* (ce qui est le cas quand la liaison a été effacée par SUPPRLI.)          */
/* Dans les conditions d'appel de FINDCH, cela revient à choisir un sommet  */
/* n'appartenant vraiment qu'à une seule liaison.                           */
/* La recherche ne se fait que sur les premiers sommets des liaisons (puis- */
/* que chaque liaison est présente dans les deux sens).                     */
/* Si un sommet est trouvé, le pointeur de la liaison dans SOMMET est       */
/* retourné. Sinon, la valeur 2*SOMSUIV est rendue.                         */
/*************************************************************************/

findextr ()
 {
  int i,fin;
  int som;
  int trouve;

  fin = 2*somsuiv;
  i = 0;
  trouve = 0;
  while ((i < fin) && (trouve == 0))
    if ((i == (fin-2)) && (sommet[i+1] != 0))
      trouve = 1;
    else if ((sommet[i] != sommet[i+2]) && (sommet[i+1] != 0))
      trouve = 1;
    else
```

```
    {
     som = sommet[i];
     i += 2;
     while ((i < fin) && (sommet[i] == som))
       i += 2;
    }
  if (trouve == 0)
    i = fin;
  return (i);
 }



/*****************************************************************************/
/* Suppression de la liaison pointée par PLI dans la table sommet. La même   */
/* liaison mais dans l'ordre différent est aussi effacée. Pour effacer une   */
/* liaison, on remplace simplement le deuxième numéro du sommet de la        */
/* liaison par 0. (On ne peut pas toucher au premier pour ne pas influencer  */
/* la recherche dichotomique.                                                */
/*****************************************************************************/

supprli (pli)
int pli;
 {
  int som1,som2;
  int i;

  som1 = sommet[pli++];
  som2 = sommet[pli];
  i = findli (som2);
  i++;
  while (sommet[i] != som1)
    i += 2;
  sommet[pli] = 0;
  sommet[i] = 0;
 }



/*****************************************************************************/
/* Recherche de toutes les chaines élémentaires de la molécule               */
/* Description de la liste de chaine :                                       */
/*   longueur de la chaine en nombre d'atome                                 */
/*   numéro du premier atome                                                 */
/*   numéro du dernier atome                                                 */
/* Chaque ensemble de chaînes liées est séparé par un enregistrement dont    */
/* les 3 valeurs sont à -1.                                                  */
/*****************************************************************************/

findch ()
 {
  int i;
  int som1,som2,length;
  int b,depile,trouve;
  int pile[maxsom],top;

  top = 0;
  convsommet ();
  trisom ();
  copyls ();
  corrcycl ();
```

```
i = findextr ();
while (i != (2*somsuiv))
 {
  length = 2;
  som1 = sommet[i];
  som2 = sommet[i+1];
  b = 1;
  while (b == 1)
   {
    while (numli[som2] == 2)
     {
      length++;
      supprli (i);
      i = findli (som2);
      som2 = sommet[i+1];
     }
    supprli (i);
    listchai[chaisuiv++] = length;
    listchai[chaisuiv++] = som1;
    listchai[chaisuiv++] = som2;
    depile = 0;
    if (numli[som2] == 1)
      depile = 1;
    else
     {
      i = findli (som2);
      if (i == 2*somsuiv)
        depile = 1;
      else
       {
        pile[++top] = som2;
        som1 = som2;
        som2 = sommet[i+1];
        length = 2;
       }
     }
    if (depile == 1)
     {
      trouve = 0;
      while ((top != 0) && (trouve == 0))
       {
        som1 = pile[top];
        i = findli(som1);
        if (i == (2*somsuiv))
          top--;
        else
         {
          trouve = 1;
          som2 = sommet[i+1];
          length = 2;
         }
       }
      if (trouve == 0)
        b = 0;
     }
   }
  listchai[chaisuiv++] = -1;
  listchai[chaisuiv++] = -1;
  listchai[chaisuiv++] = -1;
  i = findextr ();
 }
}
```

```
#include <fcntl.h>
#define maxsom 1000
#define maxcycle 1000
#define maxlm 1000
#define maxatom 1000
#define maxcoup 1000
#define maxcc 1000

int cycle [maxcycle];
int cyclsuiv;
int numcycl;
int deridcy;

int isolcycl [maxcycle/5];

int listcoup [maxcoup];
int coupsuiv;
int tbl [maxcycle];
int nbl;

int idcycle [maxcc/5*3];
int tailcyc [maxcc/5*3];
int distlicy [maxcc/5*3];
int tripcsuiv;

int listchai [maxsom/4*3];
int chaisuiv;

int idgrcy [maxcc/5];
int ref [maxcc/5];
int pc1 [maxcc/5];
int pc2 [maxcc/5];
int pr1 [maxcc/5];
int pr2 [maxcc/5];
int idgcsuiv = 0;
int ptridgr[maxcc/5];
int ptrigsuiv = 0;



/*****************************************************************************/

main ()
 {
  int i,j,k;
  int nc1,nc2,nc3;
  int tc1,tc2,tc3;
  int d,l1,l2;
  int r,p1,p2,p3,p4;
  int ptr;

  loadescr ();
  groupid ();
  i = 0;
  k = 1;
  printf ("\n");
  printf ("    Description minimale identifiante des cycles \n");
  printf ("    ---------------------------------------- \n");
  while (i < idgcsuiv)
   {
    j = ptridgr[k++];
    while (i < j)
     {
```

```
      ptr = idgrcy[i];
      nc1 = idcycle[ptr];
      tc1 = tailcyc[ptr];
      d = distlicy[ptr++];
      nc2 = idcycle[ptr];
      tc2 = tailcyc[ptr];
      l1 = distlicy[ptr++];
      nc3 = idcycle[ptr];
      tc3 = tailcyc[ptr];
      l2 = distlicy[ptr++];
      printf ("C%1d(%2d) - C%1d(%2d) - C%1d(%2d)",tc1,nc1,tc2,nc2,tc3,nc3);
      printf ("  distance: %1d  liaisons: %1d,%1d",d,l1,l2);
      r = ref[i];
      p1 = pc1[i];
      p2 = pc2[i];
      p3 = pr1[i];
      p4 = pr2[i++];
      printf ("  réf.: %2d  pos.: %1d,%1d %1d,%1d \n",r,p1,p2,p3,p4);
     }
    printf ("                              ****** \n");
   }
 }



/*************************************************************************/
/* Chargement de la description de la molécule réalisée par CODMOL.      */
/*************************************************************************/

loadescr ()
 {
  int mode,file,length;

  length = (maxcycle/5*11 + maxcoup + maxcc/5*9 + maxsom/4*3)*2 + 14;
  mode = O_RDONLY | O_RAW;
  file = open ("C:DESCRMOL.$$$",mode);
  read (file,cycle,length);
  close (file);
 }



/*************************************************************************/
/* Recherche de la liste minimale pour chaque groupe de triplets de cycles */
/* La recherche en elle-même est faite par LISTMINGR.                    */
/*************************************************************************/

groupid ()
 {
  int i,j;

  idgcsuiv = 0;
  i = 0;
  while (i < tripcsuiv)
   {
    ptridgr[ptrigsuiv++] = idgcsuiv;
    j = i;
    while (idcycle[i] != -1)
      i += 3;
    idgcsuiv = listmingr (j, i, idgcsuiv);
    i += 3;
   }
  ptridgr[ptrigsuiv] = idgcsuiv;
```

```
}



/*****************************************************************************/
/* Recherche d'une liste minimale de triplets de cycles décrivant totalement*/
/* un groupe de triplets. Cette liste doit reprendre tous les cycles du      */
/* groupe. Le premier triplet de la liste sera le premier triplet du groupe.*/
/* Un nouveau triplet est ajouté s'il contient un et un seul cycle non       */
/* encore contenu dans la liste et s'il existe dans la liste un triplet      */
/* ayant deux cycles successifs communs avec deux cycles successifs de ce    */
/* nouveau triplet.                                                          */
/* DEBGRCY pointe dans IDCYCLE le premier triplet du groupe à considérer     */
/* FINGRDY pointe le premier triplet à ne pas considérer pour ce groupe      */
/* DEBIDGRCY pointe la première valeur de IDGRCY à utiliser pour décrire la */
/*           liste minimale du groupe. IDGRCY est simplement une liste de    */
/*           pointeurs dans IDCYCLE vers les triplets retenus.               */
/* En plus de IDGRCY, on a un tableau REF qui indique pour chaque triplet    */
/* de la liste à quel autre triplet précédent il est relié. Les tableaux     */
/* PC1, PC2, PR1 et PR2 reprennent les positions des cycles communs du       */
/* triplet choisi et du triplet auquel il est relié.                         */
/* La valeur retournée est un pointeur vers la dernière valeur utilisée + 1 */
/* dans IDGRCY.                                                              */
/* Si possible, la liste minimale d'un groupe commencera par le premier      */
/* triplet de cycles non symétrique rencontré.                               */
/* Il est facile de détecter un triplet symétrique:                          */
/*   si taille du cycle central - somme des atomes communs avec les deux     */
/*   autres cycles divisée par deux (division entière) = 2 * la distance     */
/*   entre les deux cycles extrêmes, alors le triplet est symétrique.        */
/*****************************************************************************/

listmingr (debgrcy,fingrcy,debidgrcy)
int debgrcy,fingrcy,debidgrcy;
 {
  int i,j,sj;
  int c1,c2,c3,c4,c5,c6;
  int ec1,ec2,ec3,somec;
  int pi1,pi2,pj1,pj2;
  int add,trouve;
  int ok,nonsym;

  i = debgrcy;
  j = debidgrcy;
  ok = 0;
  nonsym = 2;
  while ((i < fingrcy) && (ok == 0))
    if ((tailcyc[i+1]-(distlicy[i+1]+distlicy[i+2]>>1)) == (2*distlicy[i]))
      i += 3;
    else
      ok = 1;
  if (ok == 0)
    i = debgrcy;
  idgrcy[j++] = i;
  add = 1;
  while (add == 1)
   {
    add = 0;
    i = debgrcy;
    while (i < fingrcy)
     {
      c1 = idcycle[i++];
      c2 = idcycle[i++];
      c3 = idcycle[i++];
```

```
sj = j;
j = debidgrcy;
ec1 = ec2 = ec3 = 0;
while ((j < sj) && ((ec1 == 0) || (ec2 == 0) || (ec3 == 0)))
 {
  c4 = idcycle[idgrcy[j]];
  c5 = idcycle[idgrcy[j] + 1];
  c6 = idcycle[idgrcy[j] + 2];
  j++;
  if ((c1 == c4) || (c1 == c5) || (c1 == c6))
    ec1 = 1;
  if ((c2 == c4) || (c2 == c5) || (c2 == c6))
    ec2 = 1;
  if ((c3 == c4) || (c3 == c5) || (c3 == c6))
    ec3 = 1;
 }
somec = ec1 + ec2 + ec3;
trouve = 0;
if ((ec1 == 0) && (somec == 2))
 {
  j = debidgrcy;
  while ((j < sj) && (trouve == 0))
   {
    c4 = idcycle[idgrcy[j]];
    c5 = idcycle[idgrcy[j] + 1];
    c6 = idcycle[idgrcy[j] + 2];
    j++;
    if (c2 == c5)
     {
      pi1 = 2;
      pj1 = 2;
      pi2 = 3;
      if (c3 == c4)
       {
        trouve = 1;
        pj2 = 1;
       }
      else if (c3 == c6)
       {
        trouve = 1;
        pj2 = 3;
       }
     }
    else if (c3 == c5)
     {
      pi1 = 3;
      pj1 = 2;
      pi2 = 2;
      if (c2 == c4)
       {
        trouve = 1;
        pj2 = 1;
       }
      else if (c2 == c6)
       {
        trouve = 1;
        pj2 = 3;
       }
     }
   }
 }
else if ((ec3 == 0) && (somec == 2))
 {
  j = debidgrcy;
```

```
      while ((j < sj) && (trouve == 0))
       {
        c4 = idcycle[idgrcy[j]];
        c5 = idcycle[idgrcy[j] + 1];
        c6 = idcycle[idgrcy[j] + 2];
        j++;
        if (c1 == c5)
         {
          pi1 = 1;
          pj1 = 2;
          pi2 = 2;
          if (c2 == c4)
           {
            trouve = 1;
            pj2 = 1;
           }
          else if (c2 == c6)
           {
            trouve = 1;
            pj2 = 3;
           }
         }
        else if ((c2 == c5) && ((c1 == c4) || (c1 == c6)))
         {
          pi1 = 2;
          pj1 = 2;
          pi2 = 1;
          if (c1 == c4)
           {
            trouve = 1;
            pj2 = 1;
           }
          else if (c1 == c6)
           {
            trouve = 1;
            pj2 = 3;
           }
         }
       }
      if (trouve == 1)
       {
        add = 1;
        ref[sj] = j-1;
        pc1[sj] = pi1;
        pc2[sj] = pi2;
        pr1[sj] = pj1;
        pr2[sj] = pj2;
        idgrcy[sj++] = i-3;
       }
      j = sj;
     }
   }
 return (j);
}
```

REFERENCES

1. N.L. ALLINGER, M.P. CAVA, C.R. JOHNSON, D.C. DE JONGH, N.A. LEBEL, C.L. STEVENS.
Chimie Organique.
Ed. McGraw-Hill, (1976), vol. I, chap. 1, p. 1.

2. J. MARCH.
Advanced Organic Chemistry : Reactions, Mechanisms and Structures.
Ed. McGraw-Hill, second edition, (1977), p. 1143.

3. MICHEL BOVESSSE.
Conception d'une base de données graphique en chimie organique.
Mémoire de licence, Facultés Universitaires N.D. de la Paix, Namur, (1984).

4. J.M. Trinon, J.C. GILLET, R. NOEL, T. HUWART.
Conception et réalisation d'une base de données graphique en chimie organique.
Laboratoire de deuxième licence, Facultés Universitaires N.D. de la Paix, Namur, (1986).

5. A. LEHNINGER.
Biochimie.
Ed. Flammarion Médecine-Sciences, seconde édition, (1981), chap. 11, p. 294.

6.a) B.W. KERNIGHAM, D.M. RITCHIE.
The C Programming Language.
Ed. Prentice-Hall, (1978).

b) Lattice C Manual, version 2, (mai 1982).

7.a) L.J. SCANLON.
IBM PC & XT Assembly Language : a guide for programmers.
Ed. Prentice-Hall, (1983).

b) Guide de référence de l'AMSTRAD PC 1512.
Ed. Micro Application, (1986).

8. F. CHOPLIN.
L'ordinateur en chimie.
Pour la Science, (novembre 1985), p. 50.